Visa Checkout

Integration Guide

Effective: October 31, 2018

Important Note on Copyright

Contents

Preface . . . . . **vii**

What's New in this Version vii

Chapter 1 Integration Overview **1-1**

Visa Checkout Integration Overview 1-1

About the Visa Checkout Button and Lightbox 1-1

Visa Checkout Display Language and Locale Selection 1-3

Displaying the Total Amount for a Pay Button in the Lightbox 1-5

Market Requirements 1-7

Payment Partner Reporting Requirements 1-8

About Tokenized Payment Instruments 1-8

Consumer Information Payload 1-11

Integration Steps 1-13

Integration Options 1-14

Card-on-File Transactions 1-16

About Visa Checkout Profiles 1-17

User Interface Redress Prevention 1-19

Visa's Accessibility Support 1-19

Fraud and Risk 1-20

Chapter 2 Visa Checkout Assets and Placements **2-1**

General Visa Checkout Button Placement and Flow Requirements 2-1

Chapter 3 Visa Checkout JavaScript and Button **3-1**

JavaScript Library — sdk.js 3-1

Example: JavaScript Library — sdk.js 3-1

Image Class v-button 3-1

v-button Parameters 3-2

Example: Rendering a Visa Checkout Button 3-5

Tell Me More Link 3-6

Example: Tell Me More Link 3-8

Defining onVisaCheckoutReady Function 3-8

Defining V.init Event Handler 3-9

Merchant Example 3-10

Partner Hosted Merchant Example 3-11

Payment Request Properties . . . 3-11
Settings Properties. . . . 3-16
Response to Payment Success Events . . . 3-29
Response Status . . . 3-30
Partial Shipping Address . . . 3-31
Response to Payment Cancelled Events . . . 3-32
Response to Error Events . . . 3-32
Example: Error Event Response . . . 3-32
User Data Prefill Event Handler . . . 3-33
Complete Visa Checkout Web Page HTML Example . . . 3-34
Preselected Checkout Feature . . . 3-36

Chapter 4 Mobile App Support . . . 4-1
Summary of Mobile App Options . . . 4-1
Mobile App Examples . . . 4-1
iOS Web View Hybrid App . . . 4-1
Android Web View Hybrid App . . . 4-2
Optimizing the Checkout Flow for Mobile Browsers . . . 4-2
Enabling Third-Party Cookies for Hybrid Apps . . . 4-3
Accepting Cookies in iOS Hybrid Apps . . . 4-3
Accepting Cookies in Android Hybrid Apps . . . 4-3

Chapter 5 Consumer Information . . . 5-1
About Consumer Information . . . 5-1
Consumer Information . . . 5-1
Payment Request . . . 5-3
User Data. . . . 5-4
Payment Instrument Properties . . . 5-6
Address . . . 5-14
Risk Properties . . . 5-19
3–D Secure Authentication Data Fields . . . 5-20
Wallet Info . . . 5-23
Partial Shipping Address . . . 5-24

Chapter 6 Get Payment Data. . . . 6-1
Get Payment Data Summary . . . 6-1
Get Payment Data Request . . . 6-1

Path and Endpoints 6-1
Method 6-2
Required Headers 6-3
Query Parameters 6-4
Get Payment Data Response 6-5
Full Payment Information Before Decryption. 6-6
Get Payment Data Error Response 6-8
Get Payment Data Errors 6-9
Get Payment Data Examples. 6-10
Get Summary Payment Data Success Example—Merchant 6-10
Get Summary Payment Data Success Example—Partner 6-18
Get Full Payment Data Success Example. 6-26
Get Payment Data Error Response 6-29

Chapter 7 Update Payment Info 7-1
Update Payment Info Summary 7-1
Event Types. 7-1
Card on File Events. 7-2
Promotions 7-3
Update Payment Info Request 7-3
Path and Endpoints 7-3
Method 7-3
Required Headers 7-4
Query Parameters 7-5
Update Payment Info Request Parameters. 7-5
Update Payment Info Errors 7-12
Update Payment Info Examples 7-13
Update Multiple Info Structure Examples 7-13
Order Update Success Example 7-14
Payment Update Success Example 7-16
Update Payment Info Error Examples 7-17

Chapter 8 Update Payment Info Pixel Image 8-1
Update Payment Info Pixel Image Summary 8-1
Card on File Events. 8-2
Promotions 8-2
Update Payment Info Pixel Image Request 8-2

Path and Endpoints 8-2
Update Payment Info Pixel Image Request Parameters 8-3
Update Payment Info Pixel Image Response 8-6
Update Payment Data Info Pixel Image Error Messages 8-6
Update Payment Info Request Inside an Image Tag 8-6
Update Payment Info Request 8-6
Update Payment Info Error Response 8-7

Appendix A Decrypting Consumer Information A-1
Decrypting Consumer Information Introduction A-1
Consumer Information Decryption Algorithm A-1
Consumer Information Decryption Examples A-2
Java Decryption Example A-2
C# Decryption Example. A-3
Node.js Decryption Example A-4
PHP Decryption Example A-4
Python Decryption Example A-5
Ruby Decryption Example. A-6

Appendix B HMAC-SHA256–Bit Hashing B-1
About the HMAC-SHA256–Bit Hashing Algorithm B-1
HMAC-SHA256 Hash Algorithm in PHP Example B-1
HMAC-SHA256 Hash Algorithm in Python Example. B-2
HMAC-SHA256 Hash Algorithm in Java Example B-2
HMAC-SHA256 Hash Algorithm in Ruby Example B-3
HMAC-SHA256 Hash Algorithm in C# Example B-3

Appendix C Clickjacking Prevention C-1
Clickjacking Prevention Steps C-1
Checking for Hidden Layers C-1
Using the X-Options Header C-1
Testing Your Clickjacking Prevention Implementation C-2
Example Server-Side Clickjacking Prevention Implementation C-2
Java Servlet C-2
Tomcat Configuration C-3

Appendix D AVS and CVV Responses D-1
AVS Codes D-1

CVV Codes D-3

Appendix E Branding Requirements E-1

Visa Checkout Buttons E-1

Visa Checkout Dynamic Acceptance Marks E-1

Preface

What's New in this Version

- Visa Checkout supports the use of xv2 API keys for `apikey` in JavaScript (V.init) and in API requests, which require an `x-pay-token` header. Creation of the xv2 token requires a different algorithm; however, there is no change to API key usage for making requests.

  In addition to the change of API keys, you need to provide an encryption key, `encryptionKey`, which Visa Checkout uses to encrypt data in the consumer information payload using the shared secret associated with this encryption key.
- The endpoint domains have changed to `sandbox.api.visa.com` and `api.visa.com`, for the sandbox and live, respectively.

Integration Overview 1

Visa Checkout Integration Overview

Visa Checkout is a digital payment service in which consumers can store card information for Visa, Mastercard, Discover, and American Express cards. Visa Checkout provides quick integration for merchants that want to accept payments from these card holders. Visa Checkout leverages your existing environment because most websites in which Visa Checkout will be used already exist. This means you most likely will add Visa Checkout buttons to your existing website pages and implement business and event logic using programming languages, tools, and techniques in the same way you currently do. For this reason, Visa Checkout is quite flexible and imposes very few requirements for use.

Related Content

About the Visa Checkout Button and Lightbox

Visa Checkout Display Language and Locale Selection

Displaying the Total Amount for a Pay Button in the Lightbox

Market Requirements

Payment Partner Reporting Requirements

About Tokenized Payment Instruments

Consumer Information Payload

Integration Steps

Integration Options

Card-on-File Transactions

About Visa Checkout Profiles

User Interface Redress Prevention

Visa's Accessibility Support

Fraud and Risk

About the Visa Checkout Button and Lightbox

Checking out and paying through Visa Checkout begins by proving the consumer with access to Visa Checkout from your site. This is accomplished by inserting a Visa Checkout button, which could be a generic version or one that shows an image of the credit card brand being proposed to the consumer as shown below:

- 

- 

Important

You must follow the Visa Checkout user interface guidelines.. These guidelines include information about Visa Checkout buttons, acceptance marks, and other visual assets. They also include information about integrating Visa Checkout with your checkout flow.

Regardless of how the consumer arrives at a page with a Visa Checkout button, when a consumer clicks a Visa Checkout button, one of two options appears depending on whether you use the default lightbox checkout flow or you are enabled to use the interactive checkout button flow. Both flows allow the consumer sign in, set or change billing and shipping information if desired, and make a payment.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Default Checkout Flow

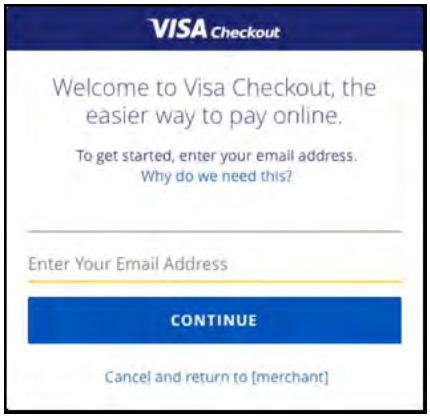
Recommended Browser Versions

Guidelines for Rendering Buttons

Z-Index Stacking

Default Checkout Flow

In the default checkout flow, the Visa Checkout lightbox appears from which the consumer can either sign up to create an account or sign in:



Related Content

About the Visa Checkout Button and Lightbox (Parent Topic)

Recommended Browser Versions

The following browsers are recommended for use by Visa Checkout:

- Internet Explorer, version 11 or later

Note

Do not use the compatibility setting; specifically, do not use it to specify a version less than IE11; for example, do not specify `x-ua-compatible=IE`, where is less than 11, in your pages.

- Firefox, current version to 10 versions prior
- Chrome, current version to 10 versions prior, excepting beta versions

- Safari, version S6 or later
- iOS, version 8 or later
- Android, version 4.4.2 or later

Other browsers may be acceptable; however, the HTML pages that contain a Visa Checkout button must be compatible with HTML 4.01 or higher, which includes XHTML 1.0 and above. Typically, you specify the HTML version in the DOCTYPE declaration for version 4.x as follows: `<!DOCTYPE html ...>`. HTML 5 does not require explicit version numbers.

Important

Visa Checkout advises consumers to upgrade to the latest version of their browser to take advantage of the latest security features that the browser offers. The browser must use Transport Layer Security (TLS) 1.2 or higher.

Related Content

About the Visa Checkout Button and Lightbox (Parent Topic)

Guidelines for Rendering Buttons

- The Visa Checkout button must be hosted by Visa.
- If any of the HTML tags that encloses the Visa Checkout button image restricts the width and height of the button, ensure that the button is rendered in its actual size.
- Do not display the Visa Checkout button as a background image.

Related Content

About the Visa Checkout Button and Lightbox (Parent Topic)

Z-Index Stacking

The Visa Checkout lightbox uses a `z-index` of `999999` to ensure it is displayed properly. Pages that host Visa Checkout buttons must use a `z-index` below `999999` to ensure that the lightbox appears on top when the consumer clicks the Visa Checkout button.

Related Content

About the Visa Checkout Button and Lightbox (Parent Topic)

Visa Checkout Display Language and Locale Selection

Visa Checkout determines the choice of language to display based on locale, which is a combination of language and country that Visa Checkout supports.

The following locales are supported since Version 2.0 unless noted otherwise:

- `es_AR` - Argentina, Spanish (Since 3.5; `en_AR` 2.9-3.4)
- `en_AU` - Australia, English
- `pt_BR` - Brazil, Portuguese (Since 3.5; `en_BR` 2.9-3.4)
- `en_CA` - Canada, English (Default for Canada)
- `fr_CA` - Canada, French
- `en_CN` - China, English (Since 2.9)

- `zh_CN` - China, Simplified Chinese (Since 3.5)

  Note

  For display purposes only; input is in English.
- `es_CL` - Chile, Spanish (Since 3.5; `en_CL` 2.9-3.4)
- `es_CO` - Colombia, Spanish (Since 3.5; `es_CO` 2.9-3.4)
- `fr_FR` - France, French (Since 4.3)
- `zh_HK` - Hong Kong, Chinese (Since 3.5)

  Note

  For display purposes only; input is in English.
- `en_HK` - Hong Kong, English (Default for Hong Kong since 2.9)
- `en_IN` - India (since 4.6)
- `en_IE` - Ireland, English (Since 4.3)
- `en_KW` - Kuwait, English (Since 5.1)
- `en_MY` - Malaysia, English (Since 2.9)
- `es_MX` - Mexico, Spanish (Since 3.5; `en_MX` 2.9-3.4)
- `en_NZ` - New Zealand English (Since 2.9)
- `es_PE` - Peru, Spanish (Since 3.5; `en_PE` 2.9-3.4)
- `pl_PL` - Poland, Polish (Since 4.3)
- `en_QA` - Qatar, English (Since 5.1)
- `en_SA` - Saudi Arabia, English (Since 5.1)
- `en_SG` - Singapore, English
- `en_ZA` - South Africa, English (Since 2.9)
- `es_ES` - Spain, Spanish (Since 4.3)
- `en_UA` - Ukraine, English (Since 5.1)
- `uk_UA` - Ukraine, Ukranian (Default for Ukraine since 5.1)

  Note

  For display purposes only; input is in English. Not all regions are supported.
- `en_AE` - United Arab Emirates, English (Since 2.9)
- `en_GB` - United Kingdom, English (Since 4.3)
- `en_US` - United States, English

This locale is one part of the algorithm that Visa Checkout uses to determine the language in which to display lightbox content. You can set a locale on each page that invokes the lightbox using the `locale` parameter, or set the locale in the merchant's profile, in which case, it applies to any page that displays the lightbox. The value of the `locale` parameter overrides any value in the merchant's profile for a page.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Language Selection Algorithm

Language Selection Algorithm

The following algorithm determines the language in which the lightbox is displayed within a browser:

Note

The rules for country selection take precedence over the rules for language selection to determine the locale, and thus, the display language.

- The country is determined by the consumer preference if changed during sign in; otherwise, it is the first item in the following hierarchy that exists or is set:
  1. Visa Checkout `country_preference` cookie in the consumer's browser
  2. Valid country in the `countryCode` setting in `V.init`
  3. Country specified in a valid locale specified in the `locale` setting in `V.init`
  4. Browser's locale

  If none of the above items can determine the country, Visa Checkout defaults to `US`. Regardless of the items listed above, after a consumer signs in, the consumer's country of registration becomes the country.
- The locale, which is used for rendering the language is determined by the consumer preference if changed during sign in; otherwise, it is the first item in the following hierarchy that exists or is set:
  1. Visa Checkout `locale` cookie in the consumer's browser
  2. Valid `locale` setting in `V.init`
  3. Default locale for the country determined from the country items above, if the resulting locale is valid for Visa Checkout
  4. Locale set by the consumer's browser / navigator

  If Visa Checkout does not support the locale, the locale becomes the default locale for the country; for example, if the locale is `fr_FR` and the country is `US`, the locale becomes `en_US`. If the locale cannot be determined by this algorithm, Visa Checkout defaults to `en_US`.

Related Content

Visa Checkout Display Language and Locale Selection (Parent Topic)

Displaying the Total Amount for a Pay Button in the Lightbox

You can specify the presentation of the total amount that is associated with a Pay button in the lightbox by using the `currencyFormat` setting of the `V.init` function. This setting affects only the presentation; it does not change the actual amount. By default, the lightbox displays a currency amount associated with the Pay button in the following format: `xxx 999,999,999.99` where `xxx` is the ISO 4217 standard alpha-3 currency code for the currency being used, suppressing leading zeros (`0`) and truncating additional precision in the display. However, the value remains unchanged.

Note

The `currencyCode` field in the `paymentRequest` of the `V.init` function determines the currency being used.

When you specify an invalid currency format, the amount is automatically displayed in the default format.

You can also set the currency format at the profile level. The `currencyFormat` field that passes in the `paymentRequest` of the `V.init` function takes precedence over the currency format value that is included in the profile. If the currency format is not specified at the profile level or in the `V.init` structure, then the total amount is automatically displayed in the default format.

You can create the `currencyFormat` string by applying the following rules:

- Specify the digits to display before and after the decimal separator by using the pound or hash mark (`#`) to indicate the presence of a digit. You must specify 9 digits to the left, a decimal separator, and 0 to 4 digits to the right of the decimal separator, such as `#########.##` Leading zeros (`0`) are always suppressed and additional precision is truncated in the display.
- Specify one of the following separator characters for the decimal separator, i.e. the decimal place, such as a decimal point or comma:
  - period (`.`)
  - comma (`,`)
  - semicolon (`;`)
  - colon (`:`)
  - space ( )
- If the decimal separator is not followed by another digit, i.e. when specifying 0 digits of precision, you must specify a period (`.`) following the right-most digit. The period will not be displayed; for example, a value of `1.00` using a format of `#########.` results in `1` being displayed.
- Optionally, specify a grouping separator character to separate values to the left of the decimal separator, such as thousands with `###,###,###.##` or `###.###.###,###`. The grouping separator character is one of the separator characters listed for the decimal place. You cannot specify a leading or tailing grouping separator.
- Optionally, to display the currency being used before or after the first or last digit, specify `currencyCode` to display the ISO 4217 standard alpha-3 code for the currency or `currencyCodeSymbol` to display the associated symbol; for example, `currencyCode ###,###,###.##` or `###,###,###.## currencyCodeSymbol`.
- The placement of the symbol or code next to the decimal formatting specifies whether a space appears before or after the number; for example, `currencyCodeSymbol###,###,###.##` for a value of `1.00` results in `$1.00`, and a format of `currencyCode ###,###,###.##` results in `USD 1.00` if the `currencyCode` field in the `paymentRequest` of the `V.init` function is `USD`. Specifying more than a single space between the symbol or code and the number is invalid and results in the default currency format being used.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Examples

Examples

`currencyFormat` String	Example of `total` 123456789.123 in United States dollars (USD)
`currencyCode ###,###,###.##`	USD 123,456,789.12
`currencyCode ###,###,###.##`	USD123.456.789,123
`currencyCodeSymbol ###,###,###.`	$ 123,456,789
`###,###,###.#### currencyCode`	123,456,789.1230 USD
`##,##,##,###.#### currencyCodeSymbol`	12,34,56,789.1230 $
`currencyCodeSymbol ###,###,###.### currencyCode`	$ 123,456,789.123 USD
`######### # currencyCode`	123456789 1USD
`currencyCodeSymbol #########.`	$ 123456789

Related Content

Displaying the Total Amount for a Pay Button in the Lightbox (Parent Topic)

Market Requirements

Visa Checkout supports billing and shipping for the following markets:

- Argentina
- Australia
- Brazil
- Canada
- China
- Chile
- Colombia
- France
- Hong Kong
- India
- Ireland

- Kuwait
- Malaysia
- Mexico
- New Zealand
- Peru
- Poland
- Qatar
- Saudi Arabia
- Singapore
- South Africa
- Spain
- Ukraine
- United Arab Emirates
- United Kingdom
- United States

Your integration must enable billing and shipping to consumers in each of these markets unless the requirement has explicitly been waived by Visa Checkout. Specifically,

1. Visa Checkout must work on merchant site/s the same way the merchant site/s accepts cards and fulfills orders paid with cards; including ***billing*** markets, ***shipping*** markets, and ***displaying*** the Visa Checkout button for every market for which the merchant supports billing.
2. Upon enablement of a new market and communication to merchants via release notes, merchants must update the integration to ensure that Visa Checkout continues working on merchant sites the same way for the new market; however, if the merchant does not support billing or shipping to the new market, the merchant is not required to support billing or shipping just for Visa Checkout.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Payment Partner Reporting Requirements

You must contact your payment partner (processor, partner, or acquirer) to inform them that you are using Visa Checkout and to determine whether there are any specific requirements from them for processing transactions with data provided by Visa Checkout. For more information, please contact your Visa Checkout representative.

Related Content

Visa Checkout Integration Overview (Parent Topic)

About Tokenized Payment Instruments

Visa Checkout supports tokenized payment instruments, which are presented to consumers as *digital account numbers*. Conceptually, a token is a replacement for the consumer's account number as it appears on a card. Tokens are considered more secure than account numbers,

which are easier to compromise and more difficult to deal with when a card associated with just an account number is lost or stolen. Tokens may reduce the risk to consumers, and to you, that the financial information associated with an account can be used for fraudulent purposes if an account number is compromised. As a result, card issuers are increasingly using tokens to authorize transactions.

To accept tokenized payment instruments:

1. Ensure that your processor can accept tokenized payment instruments.
2. Integrate to meet your processor's requirements for providing token information when authorizing a transaction.

   Note

   You also may need to support PANs if the payment instrument does not have a token replacement.

3. Request that Visa Checkout enable you to receive the token instead of a PAN when a token is available

When enabled by Visa Checkout to receive tokens, you receive consumer information payloads that contain token information if available. If the issuer provides a token for the consumer's payment instrument, you always receive the token information and do not receive account information.

If you need to know whether the payment instrument in the payload is a token or an account number before you decrypt the payload, you can determine whether it is a token or PAN by examining the `paymentMethodType` field in the `payment.success` event, which Visa Checkout returns to the frontend, or by examining the same field in the response to the Get Payment Data API. The field contains either `TOKEN` or `PAN`, depending on the type of payment instrument. The `paymentMethodType` field is also present in the payload itself.

Related Content

Visa Checkout Integration Overview (Parent Topic)

User Experience and Error Handling for Tokens

Tokenized Card on File and Multiple Authorization Considerations

User Experience and Error Handling for Tokens

Because a consumer need not be aware that a tokenized payment instrument is being used when they "choose a card," you must

- Advise the consumer to choose another card in the event of an error, regardless of whether an issue with a token-enabled payment instrument caused the error.
- Display the last 4 digits of the account number instead of the last 4 digits of the token. You can obtain the last 4 digits of the account number from the `lastFourDigits` field of the `paymentInstrument` structure in the consumer information payload.

  Note

  If you must display a date associated with the card, use 12/99 or 12/9999, because the card expiration date is not provided for tokenized payment instruments.

- Refer to tokens using the name **Digital Account Number** in cases where you must refer to a token; for example, if you display both the last 4 digits of the card as well as the last 4 digits of the token
- Not display the token expiration date. If it is necessary to display the token expiration date, it must be done in such a way that the consumer does not become confused by differing

dates; for example, display both the last 4 digits of the token and the token expiration date under **Digital Account Number** and separate token information from other account information.

Tokens cannot be auto-filled/key-entered in the browser. You should work with your Visa Checkout representative to develop consumer-facing messaging about tokens, as needed.

Related Content

About Tokenized Payment Instruments (Parent Topic)

Tokenized Card on File and Multiple Authorization Considerations

You can use tokens for Card on File type transactions. The merchant stores the call ID for the transaction and makes a Get Payment Data API request to obtain a consumer's information payload which includes the token information.

Note

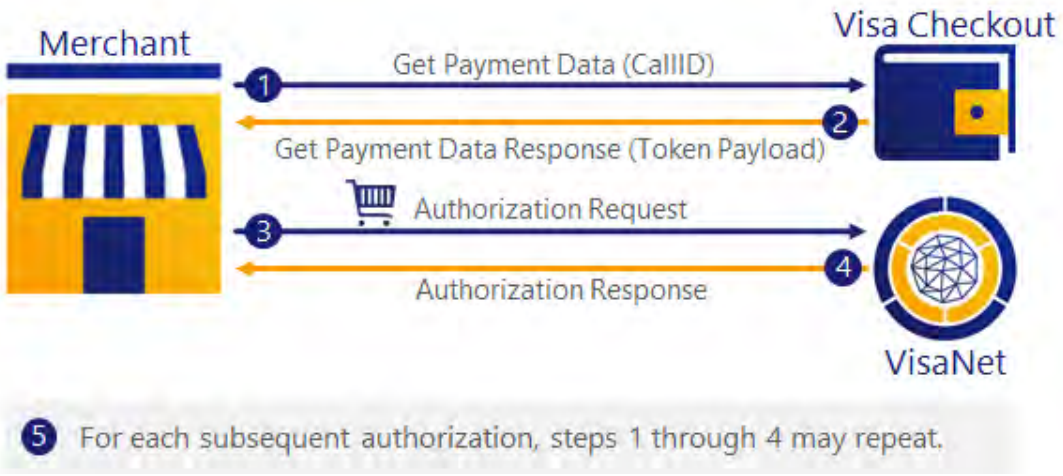
Visa Checkout supports tokenized payments instruments for Visa cards only.

The Get Payment Data response includes a new cryptogram for the token, which can be used for a subsequent authorization when transactions require multiple authorizations. Because the Get Payment Data API response returns a new cryptogram, you should make a Get Payment Data API request to refresh the cryptogram prior to using the token each time the token is needed for a subsequent authorization.

Note

If you store the call ID, do not request an expiration for the call ID.

The following diagram shows how multiple authorizations work with tokens:



The token information appears in the `paymentInstrument` structure of the payload that Visa Checkout returns. It includes the `tokenInfo` and `cryptogramInfo` elements.

Consumers do not need to be aware a token has replaced their credit card number. Also, tokens cannot be entered into a browser or autofilled.

The following steps are shown:

1. Call Get Payment Data to obtain the token information.
2. The response includes a new cryptogram, which you must use for a subsequent authorization.

3. Pass the token information, including the new cryptogram to your processor as part of your authorization request.

   Note

   You follow your processor's protocol for requesting an authorization. Your processor typically handles VisaNet operations.

4. The processor returns the status of the authorization using their protocol. If the authorization fails, you must contact the consumer to obtain another payment instrument.
5. You can repeat these steps for additional authorization requests. You cannot skip any of these steps because each Get Payment Data response includes a new cryptogram, which must be passed with the token to your processor.

Related Content

About Tokenized Payment Instruments (Parent Topic)

Consumer Information Payload

On successful completion of a payment request, Visa Checkout returns a payload of consumer information, depending on how your account has been configured by Visa Checkout or your Visa Checkout partner when your account was created. The consumer information contains information about the following:

- Consumer, such as the consumer's name and email address
- Payment request, such as the payment amount and currency, shipping and handling charges, discounts, promotion codes, and such
- Payment instrument, such as a PAN or token and related information, billing address, card art, and such
- Risk information, such as AVS and CVV responses
- Shipping address if requested
- Verified by Visa authentication information, if configured by Visa Checkout

Important

The payload can contain *personally identifiable information* (PII). You must follow PCI compliance guidelines when dealing with PII.

How the account was created and the requested data level determine whether the payload contains the consumer's account number (PAN) or token, which represents the account number in tokenized payment instruments. Specifically, if you have been enabled to receive PAN access, which includes token access, you receive the full encrypted consumer payload; otherwise, you receive encrypted summary information, meaning no PAN or token plus associated cryptogram, regardless of whether you requested full or summary information.

The consumer information payload is always encrypted when returned to your website, regardless of whether it contains full or summary information. When requested by a Visa Checkout API, full information is always encrypted; however, summary information is not encrypted.

Note

You can choose not to receive the payload as part of a successful payment request; however, you must then call a Visa Checkout API when you are ready to obtain it. When you enable the consumer information payload to be returned directly, which is also the default, an API-based integration may not be necessary.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Token Available—tokenInfo Structure Returned in Payload

Account Number (PAN) Available—No tokenInfo Structure Returned in Payload

Token Available—tokenInfo Structure Returned in Payload

Token information appears in the `paymentInstrument` structure of the payload. It consists of 2 structures, `tokenInfo` and `cryptogramInfo`. The contents of the `tokenInfo` structure depend on whether PAN access, which also applies to tokens, has been granted and whether full or summary data has been requested.

Fields returned when PAN access is enabled and full data requested	**Fields returned when PAN access is not enabled or summary data requested**
Payment method type (`TOKEN`)	Payload contains a `tokenInfo` structure
Token	–
Token range	Token range
Token, last 4 digits	Token, last 4 digits
Token expiration date	Token expiration date
Cryptogram	–
ECI	–
Payment Account Reference (PAR)	Payment Account Reference (PAR)

Note

The last four digits of the underlying PAN is also available for display purposes.

Related Content

Consumer Information Payload (Parent Topic)

Account Number (PAN) Available—No tokenInfo Structure Returned in Payload

Fields returned when PAN access is enabled and full data requested	**Fields returned when PAN access is not enabled or summary data requested**
Payment method type (PAN)	Payment method type (PAN)
Account number (PAN)	–
Last four digits	Last four digits

Fields returned when PAN access is enabled and full data requested	Fields returned when PAN access is not enabled or summary data requested
Card BIN, 6 digits	Card BIN, 6 digits
Card expiration Date, Month and Year	Card Expiration Date, Month and Year

Related Content

Consumer Information Payload (Parent Topic)

Integration Steps

At a high level, your integration consists of modifying your checkout and payment pages to:

1. Place the Visa Checkout button on your page and provide JavaScript to enable it:
   - Load the Visa Checkout JavaScript library, `sdk.js`.
   - Initialize the library properties related to the lightbox appearance and the payment request.
   - Provide event handlers that respond when the lightbox closes, including a *payment success* handler to set up payment processing using your own business logic.
2. Decrypt the payload returned with a *payment success* event and process the payment
3. Update payment information in Visa Checkout after the payment has been processed

All integrations require you to perform Step 1. Visa Checkout provides you with the button to use; however, there is considerable flexibility for its use. See Getting Started With Visa Checkout for button placement and usage information. You must implement 1 JavaScript function to specify library properties and implement handlers for lightbox events.

In Step 2, a payment success event returns encrypted consumer payment information, which includes card verification, authentication, and risk information. The account number (PAN) can be returned by agreement with Visa Checkout. Typically, you use your existing business logic to process the payment request. Whether you need to decrypt or use this information depends on your business logic and who performs it.

Note

In addition to a payment success event, you must also handle

- A payment cancellation event, which indicates that the consumer closed the lightbox before confirming the payment request.
- A payment error event, which indicates that an error occurred during the operation of the lightbox, which in most cases indicates an issue with the payment request or initialization of the lightbox.

How you update payment information in Visa Checkout (Step 3) depends on your existing business logic, current capabilities and security requirements. In some cases, requirements may be imposed by your processor or an e commerce partner, which is someone you might choose to handle Visa Checkout transactions on your behalf. Visa Checkout provides several integration options to meet your requirements, which are described in Updating Payment information in Visa Checkout.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Integration Options

Visa Checkout provides several options to manage consumer payment information that is returned by a payment event. It automatically updates Visa Checkout as a result of having processed a consumer's payment. You can manage these tasks by:

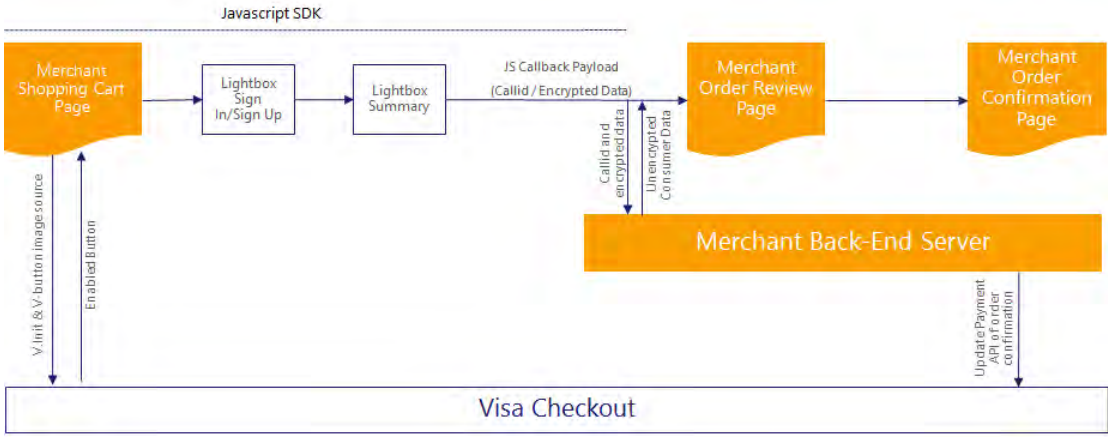
- Taking action from your web page or front-end server
- Calling a Visa Checkout API from either a front-end or back-end server.
- Passing the ID that is associated with the Visa Checkout payment request, which is represented as a *call ID* in Visa Checkout terminology, to your server or to a processor or an e-commerce partner for payment processing

Note

You can pass a call ID to a server as a convenience for remote process communication.

These choices are not mutually exclusive; for example, you can process consumer payment information with your front-end server and update payment information in Visa Checkout another way.

The following illustration demonstrates one way a merchant can integrate Visa Checkout:



Related Content

Visa Checkout Integration Overview (Parent Topic)

Responding to Payment Events

Updating Payment Information in Visa Checkout

Visa Checkout API Summary

Responding to Payment Events

A payment event occurs when the consumer completes the payment request, the consumer cancels the request, or an error occurs while the lightbox is open. If the payment event indicates success, consumer information is available to complete the payment, What you do with the payment information depends on what information you need and how you complete the payment. The consumer payment information is encrypted in the payload returned with the event.

Note

YOU ARE RESPONSIBLE FOR THE SECURITY OF THE INFORMATION BEING DECRYPTED. NEVER DECRYPT THE PAYLOAD DIRECTLY IN YOUR WEB PAGE.

The transaction's call ID is provided along with the event. You have three choices for handling the event:

- Pass the call ID to your server for payment processing or to the entity that will process the payment for you; in which case, the call ID can be used with the Get Payment Data API to obtain the consumer payment information.
- Pass the encrypted payload to your server for payment processing or to the entity that will process the payment for you.
- Call a Visa Checkout API, Get Payment Data, to obtain the consumer payment information from the server you use to handle payment processing, This is what an entity that processes payments on your behalf must do also.

Related Content

Integration Options (Parent Topic)

Updating Payment Information in Visa Checkout

After the payment has been processed, you must update Visa Checkout with the information. This might occur in real time, immediately after the lightbox closes, or could happen later. You may take the action yourself or it might be handled on your behalf by a payment processor or an e-commerce partner. The following integration options are available, depending on your configuration:

- Update Visa Checkout payment information from your web page by passing parameters when the 1-pixel image provided by Visa Checkout is loaded.
- Pass the call ID to your server, or to the entity that will take action to on your behalf. In this case, the call ID can be used with the Update Payment Info API to update Visa Checkout payment information.
- Call the Visa Checkout API, Update Payment Info, to update payment information.

Related Content

Integration Options (Parent Topic)

Visa Checkout API Summary

The Visa Checkout API consists of REST-style messages, whose request and response pairs are transported using the HTTPS protocol. Many programming languages provide HTTPS interfaces, or you can send requests and receive responses as text; of course, there are headers and encryption involved. Visa Checkout does not dictate the use of any particular programming language to use its APIs.

Note

For maximum compatibility with Visa Checkout services, use Get Payment Data and Update Payment Info APIs.

If you choose to use Visa Checkout APIs, you must support DNS name resolution. A list of static IP addresses cannot be provided.

The following APIs are available to all merchants:

Field	Description
payment/data/{callId}	GET obtains consumer payment information associated with the payment request (callId). It provides the same information as though you used the Visa Checkout JavaScript library.
payment/data/{callId}	PUT updates the status of the transaction and the amounts associated with the payment request (callId) specified in the Visa Checkout library initialization. It is an alternative to using the Update Payment Information pixel image on your web pages.

Related Content

Integration Options (Parent Topic)

Card-on-File Transactions

In Visa Checkout, a card-on-file transaction is one in which the merchant initiates a transaction using previously collected card information. You must have permission from Visa Checkout to create card-on-file transactions.

A Cardholder-initiated Transaction (CIT) is any transaction where the cardholder actively participates in the transaction. This transaction can be at a terminal in a store, through a checkout experience online, or with a stored payment credential that the cardholder has previously consented to store with the merchant.

A Merchant Initiated Transaction (MIT), also known as a card-on-file transaction, is any transaction that relates to a previous cardholder-initiated transaction but is conducted without the active participation of the cardholder. The MIT transaction uses a stored credential and represents the cardholder agreement for the merchant to initiate one or more future transactions over a period for a single purchase of goods or services.

Merchants commonly perform MITs to perform a:

- Transaction as a follow-up to a cardholder-initiated transaction
- Pre-agreed standing instruction from the cardholder for the provision of goods or services

Examples of MITs include a:

- Hotel charge for mini-bar expenses tallied after the guest has checked out and closed the folio
- Subsequent recurring payment for a magazine subscription

You typically confirm all transactions by calling the Update Payment Info API.

When a consumer checks out using the lightbox during the initial transaction, a call ID is returned with the consumer information payload. For the initial transaction, use the initial call ID and specify Confirm for the event type. For subsequent card-on-file transactions, the consumer does not invoke the lightbox. You use the same call ID and specify Confirm_COF for the event type.

Note

If you want to place a card-on-file without the consumer making a purchase, use Create for the event type.

Merchants who save the `callId` on file and invoke the Get Payment Data API, receive the latest PAN updates that the Visa Account Updater (VAU) has completed.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Automatic Updates to Card-on File in Visa Account Updater

Automatic Updates to Card-on File in Visa Account Updater

Visa Account Updater (VAU) is a service that is designed to address the requirements of recurring payments and episodic transactions. It enables the secure exchange of updated payment account information among participating issuers, acquirers, and qualified account-on-file merchants.

VAU supports account renewal or card replacement; account upgrades or downgrades; portfolio acquisitions and/or mergers; lost/stolen cards; other account closures; and MasterCard-to-Visa conversions. Visa Checkout uses VAU update information to ensure that a user's card information is up to date and that point-of-sale disruptions due to stale account information are minimized.

The VAU update payment account information is now available with Visa Checkout. Issuers who receive standard VAU reports can view **VISACHECKOUT**. Visa Checkout starts processing additional VAU data for **Closed Account** and **Please Call update** types. As existing backlogs are cleared, issuers may notice a short-term increase in the number of cards that have been deleted from Visa Checkout.

When the card issuer is enrolled in Visa Account Updater (VAU) process, the process can update the following information in Visa Checkout:

- Expiration date, when the issuer provides a new expiration date; typically, when the card expires
- Account number, when the issuer provides a replacement account number for a number that is no longer valid
- **Closed Account** type
- **Please Call update** type

All other information associated with the card remains unchanged as a result of this process, which runs daily.

Note

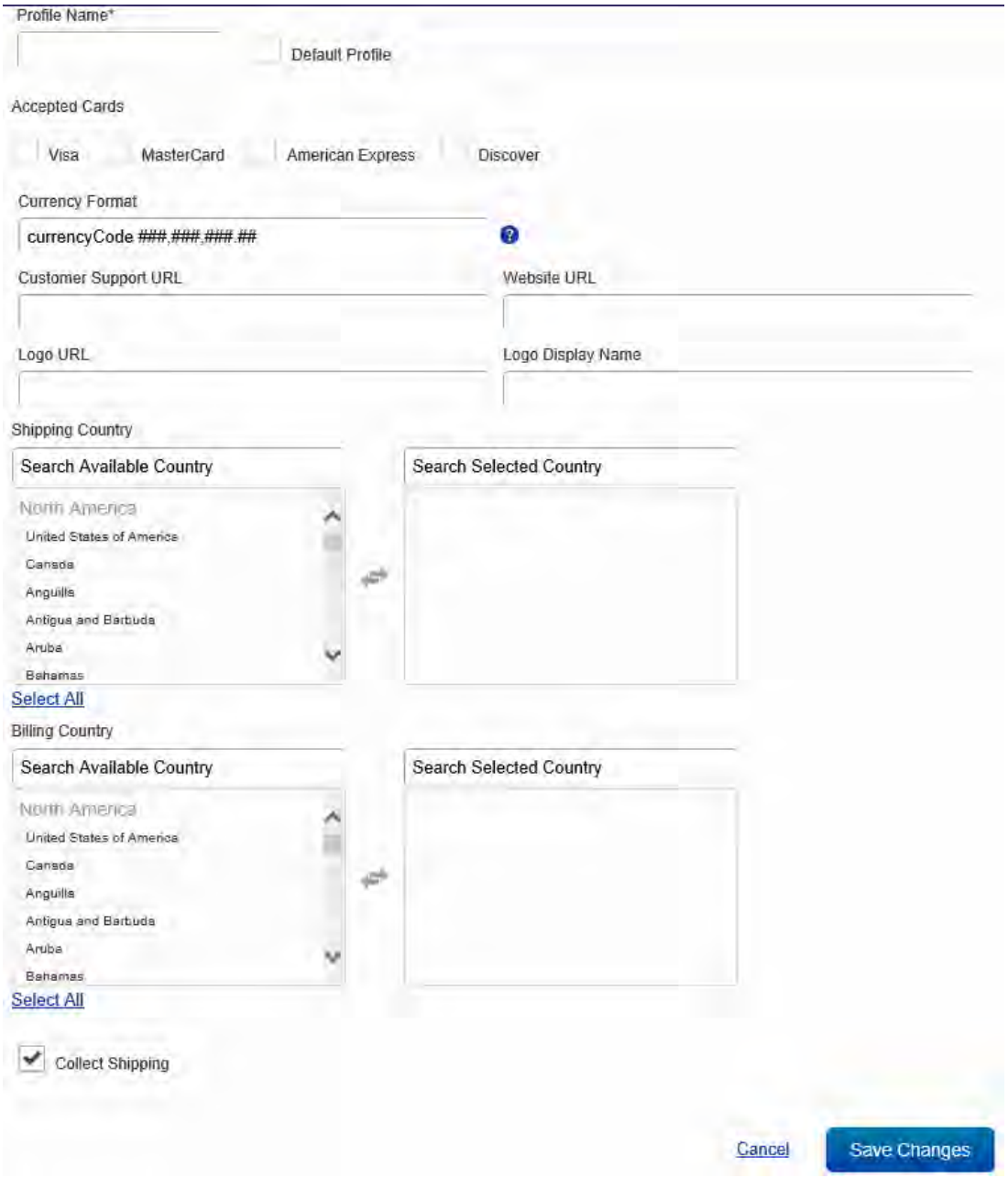
The consumers of a card must update the card expiration date or account number themselves if the card issuer is not enrolled in the VAU process or does not update card information on a timely basis.

Related Content

Card-on-File Transactions (Parent Topic)

About Visa Checkout Profiles

All merchants should have a default Visa Checkout profile established that specifies the lightbox settings your consumers will see, such as which cards are accepted, which billing countries are accepted (which in turn, affects currencies), which countries shipping is accepted, and a custom-tailoring of the lightbox display. When you log into the Visa Checkout Developer Center, you can select **Profile** from the navigation on the left and complete the following fields:



Note

If you have an e-commerce partner, your partner may provide a profile for you to use. Also, in some situations, your e-commerce partner may provide an alternative to using the Visa Developer Center, in which case, you may not be able to log into the Developer Center. In those situations, you should follow your e-commerce partner's instructions.

Visa Checkout will use the settings in the default profile unless you either:

- Specify a value for a setting using the JavaScript SDK. These are `V.init` settings.
- Specify another profile to use, which is specified in the `externalProfileId` setting in `V.init`.

The order in which Visa Checkout selects values for `V.init` settings is as follows:

1. The value specified in `V.init`

2. The value specified in the selected (`externalProfileId`) profile
3. The value specified in the default profile
4. The default Visa Checkout value, which is the default `V.init` value, assuming no profile options have been set

Related Content

Visa Checkout Integration Overview (Parent Topic)

User Interface Redress Prevention

It is possible for malicious code to monitor consumers' keystrokes in an attempt to steal confidential information. For example, when consumers believe they are selecting legitimate links on an infected web page, they are actually selecting a transparent layer over the links that relays information to outside unauthorized sources; it also applies to buttons and other UI controls. This practice is referred to as *clickjacking*. You must ensure that each page containing a Visa Checkout button has adequate anti-clickjacking preventions in place.

To prevent clickjacking, which could occur if malicious code is hidden beneath legitimate buttons or other clickable content on your web page, you must:

- Provide anti-clickjacking code, which is typically JavaScript in the header of each page that hosts a Visa Checkout button, to ensure that the associated DOM document for the page has no child pages in which malicious code could reside.
- Provide headers on your server to prevent your page from being loaded from another domain. For example, implement `X-FRAME-OPTIONS DENY` or `X-FRAME-OPTIONS SAMEORIGIN` filtering for headers on your server to handle this requirement and provide Content-Security-Policy frame ancestors for browsers that support them.

  ### Important

  You must implement both measures (code and headers) to help ensure that pages cannot be loaded as an iFrame of some other page.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Visa's Accessibility Support

Visa has a robust process for coding to and validating conformance to WCAG 2.0 AA (the specification developed by W3C). For each project that includes accessibility in their requirements, Visa's accessibility team (within Visa User Experience) holds an accessibility kickoff/introductory meeting for involved staff to review the standards and Visa's process. Staff then take a 90–minute online training class that trains them in accessibility, WCAG 2.0, and how to use Visa's Global Accessibility Requirements (VGAR). The VGAR is a searchable web portal containing a set of (currently) 120 specific requirements that Visa projects must follow to meet WCAG 2.0 (meeting the VGAR means a project meets WCAG 2.0 AA) and is hosted inside Visa's network. The VGAR also includes a re-sort of the requirements and specific test cases for each project's QA staff, along with downloadable tracking tools, links to all necessary testing tools, and how-to videos for every tool and test case. Throughout the process if Development or QA has any questions, they can reach out to Visa a11y (accessibility) for consulting. Visa is committed to working with our merchants and partners to satisfy their accessibility compliance needs.

Related Content

Visa Checkout Integration Overview (Parent Topic)

Fraud and Risk

Related Content

Visa Checkout Integration Overview (Parent Topic)

Fraud Checks

Declines

Card Security Code Usage

Verified by Visa (3–D Secure) Transactions

Fraud Checks

Visa Checkout uses a combination of proprietary and third-party technologies to implement fraud checks during the consumer checkout experience in Visa Checkout. These checks provide account validations on all Visa Checkout accounts when:

- The account is added or accessed
- A consumer logs in to Visa Checkout
- A card is associated with the account, a card is updated, or used in a transaction
- When a consumer makes changes to the account

Examples of fraud checks include device and IP data checks, velocity, card and account verification, enrollment attributes, Visa Checkout transaction history, and other checks. Specifically, for every card added to a Visa Checkout account, regardless of card brand, Visa Checkout performs an account verification procedure prior to passing the card information to a merchant.

Important

Although Visa Checkout performs an array of proprietary fraud checks while interacting with consumers, Visa Checkout never declines a transaction request based on risk concerns. Your own control models, processes, and procedures should provide the best protection against fraud based on the philosophy that you know your customers and their behavior the best and are in the best position to assess your own risk tolerance for a given transaction. **VISA CHECKOUT FRAUD CHECKS SHOULD NEVER BE USED TO REPLACE OR SUPPLANT YOUR OWN TECHNIQUES; RATHER, THEY SHOULD SUPPLEMENT YOUR EXISTING CONTROLS.**

Related Content

Fraud and Risk (Parent Topic)

Declines

Declines are the responsibility of the card issuer and the merchant. Visa Checkout does not decline transactions at a transaction level, except in extreme circumstances; for example, when an account has been disabled due to suspicious activity or a government sanctions list match.

Related Content

Fraud and Risk (Parent Topic)

Card Security Code Usage

Visa Checkout performs a verification of the card security code for each card added to a Visa Checkout transaction. Similar to a card-on-file scenario, the validation is performed once, without re-verifying the card security code during each use of the card.

Important

Never collect from consumers their CVV2, CVN, CVC2, CID or any other such security feature for any “card not present” transactions (collectively called card security codes) separate and apart from Visa’s collection of the same via the checkout experience with the Visa Checkout Services unless you have Visa’s express written consent to do so, or your collection of the card security code is specifically required by Visa's Rules. You must never store card security codes.

You are encouraged to implement best practices in regard to risk management for Visa Checkout transactions as you would for any other e-commerce transaction. Because a card security code has been validated for the Visa Checkout payment method being used, a historical match response should be assumed.

Currently, card brands supported by Visa Checkout do not downgrade interchange based on the absence of a card security code for "card not present" transactions. You should check with your acquirer or processor to determine whether they have any policies or fees specific to your contract that may be related to authorizations that do not contain a card security code.

Typically, the card security code in a response is optional information that can be included in a re-presentment. However, whether a card security code is required to reverse a particular charge-back, may depend on the card brand. Merchants are encouraged to speak directly with their acquirer to understand the charge-back re-presentment rules and reversal criteria for a specific card brand.

Note

Although AVS and card security codes, e.g. CVV2, are 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 those values.

Related Content

Fraud and Risk (Parent Topic)

Verified by Visa (3–D Secure) Transactions

When configured, Visa Checkout supports Verified by Visa (VbV) authentication checks and returns VbV authentication information in the consumer information payload. You will provide VbV information in the payload to your processor as required by the processor's authorization message. Contact your Visa representative for VbV specific configuration information.

Related Content

Fraud and Risk (Parent Topic)

Enabling Strong Authentication on First Use of a Card

Enabling Strong Authentication on First Use of a Card

When configured for 3-D Secure, authentication generally takes place on each transaction within the Visa Checkout experience. In each of the following designated countries, however, merchants and partners must allow issuers to authenticate consumers on the first use of the card in Visa Checkout:

- France

- India
- Ireland
- Poland
- Spain
- United Kingdom

Visa Checkout supports authentication on first use of Visa, Mastercard, and American Express cards in these designated countries. Merchants and partners can configure their Visa Checkout account to use 3-D Secure to perform a consumer authentication check when Visa, Mastercard, and American Express cards are first used in Visa Checkout in these designated countries.

Merchants or partners can configure 3-D Secure for Visa, Mastercard, and American Express in Visa Checkout to apply on first use only. Alternatively, merchants or partners can configure 3–D Secure for Visa, Mastercard, and American Express for every transaction only. Merchants and partners can also configure 3–D Secure for Visa, Mastercard, and American Express in Visa Checkout on first use and for every transaction. The Suppress Challenge override (`threeDSSuppressChallenge`) setting is ignored and the issuer authentication step-up, if available, is always presented.

When configured to authenticate on first use of a card, authentication fields are returned in the consumer information payload when a consumer in a designated country first uses a card in Visa Checkout, even when 3-D Secure has not been activated for general use.

Important

If merchants or partners do not have a Visa Checkout profile for 3-D Secure configured to enable the required consumer authentication during a first Visa Checkout transaction, or if the authentication fails, the merchant or partner must perform the required consumer authentication in their own checkout experience.

When consumer authentication is performed on first use, the `firstUseAuthenticated` field is returned in the consumer information payload.

Related Content

Verified by Visa (3–D Secure) Transactions (Parent Topic)

Visa Checkout Assets and Placements 2

General Visa Checkout Button Placement and Flow Requirements

You are required to implement the Visa Checkout branding requirements on all pages where the consumer is presented payment method options, such as Visa Checkout or another payment method. Common examples include shopping cart pages, login pages, product pages, and payment pages. Your actual implementation depends on your specific flow.

You can use Visa Checkout on any page or in any flow on your site or app where a consumer is asked to enter their billing and payment information. Common examples include cart pages (both full and mini) pages, payment pages, card-on-file management pages, or immediately before a flow where a consumer is prompted for personal information, which may be available, at least partially, within Visa Checkout.

Because Visa Checkout already has consumer shipping information and payment options, giving consumers the opportunity to specify choices at the beginning of the checkout process may enable them to complete the transaction with less effort that might otherwise be required. The following diagram show how placing Visa Checkout buttons on a shopping cart and log in page might work:

Some consumers may not select the Visa Checkout button initially, in which case you should also offer Visa Checkout to consumers when they choose a payment method, which still enables them to use Visa Checkout for payment:

The implementation flow above shows a radio button with a Visa acceptance mark. When the consumer chooses the Visa Checkout radio button, the Visa Checkout button appears, which enables the consumer to log in to Visa Checkout to choose their payment card before continuing on to your payment page.

Visa Checkout JavaScript and Button 3

JavaScript Library — sdk.js

Use the `sdk.js` JavaScript library to control the operation of Visa Checkout on your site.

Sandbox Path

```
https://sandbox-assets.secure.checkout.visa.com/checkout-widget/resources
/js/integration/v1/sdk.js
```

Live Path

```
https://assets.secure.checkout.visa.com/checkout-widget/resources
/js/integration/v1/sdk.js
```

Related Content

Example: JavaScript Library — sdk.js

Example: JavaScript Library — sdk.js

```
<body>
 ...
 <script type="text/javascript"
  src="https://sandbox-assets.secure.checkout.visa.com/
  checkout-widget/resources/js/integration/v1/sdk.js">
 </script>
</body>
```

Related Content

JavaScript Library — sdk.js (Parent Topic)

Image Class v-button

Use `v-button img` class to render a Visa Checkout button on your web site to initiate a payment. Rendered buttons must follow Visa Checkout user interface guidelines.

Sandbox Path

```
https://sandbox.secure.checkout.visa.com/wallet-services-web/xo/button.png
```

Live Path

```
https://secure.checkout.visa.com/wallet-services-web/xo/button.png
```

Related Content

v-button Parameters

Example: Rendering a Visa Checkout Button

v-button Parameters

Field	Description
`size`	*(Optional)* Width of the button, in pixels. You can either specify `size` to display a standard size button, or you can specify `height` and `width` to specify a custom size. If you do not specify `size` or both `height` and `width`, the button size is 213 pixels. If you specify `height` or `width`, the value of `size` is ignored. **Format:** It is one of the following values: • `154` - small • `213` - medium (default) • `425` - High resolution or large Any other value defaults to 213 pixels. **Example:** `size=154` Since 2.0
`height`	Height of the button, in pixels, for custom button sizes. You must specify the height if you specify a value for `width`. The value you choose determines the range of allowable values for `width`. **Format:** It is one of the following values: • `34` • `47` • `94` **Example:** `height=94` Since 2.4
`width`	Width of the button, in pixels, for custom button sizes. You must specify the width if you specify a value for `height`. **Format:** It is one of the following values: • less than `477` if `height` is `34`; default value is `154` • greater than `212` and less than `659` if `height` is `47`; default value is `213`

Field	Description
	• greater than `424` and less than `1317` if `height` is `94`; default value is `425` The default value is used if the value for `width` is invalid for the specified height. **Example:** `width=200` Since 2.4
`locale`	*(Optional)* The locale, which controls how text displays in a Visa Checkout button and the Visa Checkout lightbox. **Format:** It is one of the following values: • `es_AR` - Argentina, Spanish (Since 3.5; `en_AR` 2.9-3.4) • `en_AU` - Australia, English • `pt_BR` - Brazil, Portuguese (Since 3.5; `en_BR` 2.9-3.4) • `en_CA` - Canada, English • `fr_CA` - Canada, French • `en_CN` - China, English (Since 2.9) • `zh_CN` - China, Simplified Chinese (Since 3.5) **Note** For display purposes only; input is in English. • `es_CL` - Chile, Spanish (Since 3.5; `en_CL` 2.9-3.4) • `es_CO` - Colombia, Spanish (Since 3.5; `es_CO` 2.9-3.4) • `fr_FR` - France, French (Since 4.3) • `zh_HK` - Hong Kong, Chinese (Since 3.5) **Note** For display purposes only; input is in English. • `en_HK` - Hong Kong, English (Since 2.9) • `en_IN` - India, English (Since 4.6) • `en_IE` - Ireland, English (Since 4.3) • `en_KW` - Kuwait, English (Since 5.1) • `en_MY` - Malaysia, English (Since 2.9) • `es_MX` - Mexico, Spanish (Since 3.5; `en_MX` 2.9-3.4) • `en_NZ` - New Zealand English (Since 2.9) • `es_PE` - Peru, Spanish (Since 3.5; `en_PE` 2.9-3.4)

Field	Description
	• pl_PL - Poland, Polish (Since 4.3) • en_QA - Qatar, English (Since 5.1) • en_SA - Saudi Arabia, English (Since 5.1) • en_SG - Singapore, English • en_ZA - South Africa, English (Since 2.9) • es_ES - Spain, Spanish (Since 4.3) • en_UA - Ukraine, English (Since 5.1) • uk_UA - Ukraine, Ukranian (Since 5.1) **Note** For display purposes only; input is in English. • en_AE - United Arab Emirates, English (Since 2.9) • en_GB - United Kingdom, English (Since 4.3) • en_US - United States, English Since 2.0
color	*(Optional)* The color of the Visa Checkout button. **Format:** It is one of the following values: • standard (default)  • neutral  **Note** Any other value for color will default to standard **Example:** color=neutral Since 2.5
cardBrands	*(Optional)* Override value for brands associated with card art to be displayed. If a brand matching the consumer's preferred card is specified, the card art is displayed on the button; otherwise, a generic button is displayed. VISA must be included if acceptCanadianVisaDebit is true. **Format:** Comma-separated list of one or more of the following brands: • VISA • MASTERCARD

Field	Description
	• AMEX • DISCOVER • ELECTRON (Brazil only; since 3.9) • ELO (Brazil only; since 3.9) **Example:** cardBrands=VISA,AMEX Since 2.0
acceptCanadianVisaDebit	Whether a Canadian merchant accepts Visa Canada debit cards; required for Canadian merchants, otherwise, ignored. Visa must be specified as an allowable card brand. **Format:** It is one of the following values: • true - Visa Canada debit cards accepted • false - Visa Canada debit cards not accepted **Example:** acceptCanadianVisaDebit="true" Since 2.0
cobrand	For future use. **Format:** It is one of the following values: • true • false **Example:** cobrand="true" Since 4.6

Related Content

Image Class v-button (Parent Topic)

Example: Rendering a Visa Checkout Button

```
<body>
 ...
 <img alt="..." class="v-button" role="button" src=
"https://sandbox.secure.checkout.visa.com/wallet-services-web/xo/button.png?..."
/>
...
 </body>
```

Note

You can specify tabbing behavior to the button by including the tabindex attribute:

```
<img alt="..." class="v-button" role="button" tabindex="0" src=
"https://sandbox.secure.checkout.visa.com/wallet-services-web/xo/button.png?..."
/>
```

Related Content

Image Class v-button (Parent Topic)

Tell Me More Link

Use the `v-learn` `<a>` (hyperlink) class to provide a **Tell Me More** link that a consumer clicks to learn more about Visa Checkout. The class causes a pop up to be displayed in the specified language, which by default is `en_US`. An example link and associated pop-up window are shown in *Getting Started With Visa Checkout*.

You must provide the link's text, which typically is **Tell Me More**, in the specified locale:

Field	Description
`data-locale`	*(Optional)* The locale, which controls how the pop up text displays in a Tell Me More link. **Format:** It is one of the following values: • `es_AR` - Argentina, Spanish (Since 3.5; `en_AR` 2.9-3.4) • `en_AU` - Australia, English • `pt_BR` - Brazil, Portuguese (Since 3.5; `en_BR` 2.9-3.4) • `en_CA` - Canada, English • `fr_CA` - Canada, French • `en_CN` - China, English (Since 2.9) • `zh_CN` - China, Simplified Chinese (Since 3.5) **Note** For display purposes only; input is in English. • `es_CL` - Chile, Spanish (Since 3.5; `en_CL` 2.9-3.4) • `es_CO` - Colombia, Spanish (Since 3.5; `es_CO` 2.9-3.4) • `fr_FR` - France, French (Since 4.3) • `zh_HK` - Hong Kong, Chinese (Since 3.5) **Note** For display purposes only; input is in English. • `en_HK` - Hong Kong, English (Since 2.9) • `en_IN` - India, English (Since 4.6) • `en_IE` - , English (Since 4.3) • `en_KW` - Kuwait, English (Since 5.1) • `en_MY` - Malaysia, English (Since 2.9) • `es_MX` - Mexico, Spanish (Since 3.5; `en_MX` 2.9-3.4) • `en_NZ` - New Zealand English (Since 2.9) • `es_PE` - Peru, Spanish (Since 3.5; `en_PE` 2.9-3.4) • `pl_PL` - Poland, Polish (Since 4.3) • `en_QA` - Qatar, English (Since 5.1) • `en_SA` - Saudi Arabia, English (Since 5.1) • `en_SG` - Singapore, English • `en_ZA` - South Africa, English (Since 2.9) • `es_ES` - Spain, Spanish (Since 4.3)

Field	Description
	• en_UA - Ukraine, English (Since 5.1) • uk_UA - Ukraine, Ukranian (Since 5.1) **Note** For display purposes only; input is in English. • en_AE - United Arab Emirates, English (Since 2.9) • en_GB - United Kingdom, English (Since 4.3) • en_US - United States, English Since 2.5

The v-learn class does not provide styling. To provide default styling, you can use the v-learn-default class in addition to the v-learn class. The v-learn-default class provides default styling, e.g., color, font, size, and right-alignment of the text to the Visa Checkout button's container, not to the button itself. The Visa Checkout button and Tell Me More link must to be wrapped inside a parent , whose width is the width of the button.

Related Content

Example: Tell Me More Link

Example: Tell Me More Link

The following example shows how to provide a **Tell Me More** link with default styling:

```
<div class="v-checkout-wrapper">
 <img
  class="v-button" role="button" alt="Visa Checkout"
  src="https://sandbox.secure.checkout.visa.com/wallet-services-web/xo/button.png">
 <a class="v-learn v-learn-default" href="#" data-locale="en_US">Tell Me More</a>
</div>
```

Related Content

Tell Me More Link (Parent Topic)

Defining onVisaCheckoutReady Function

The Visa Checkout button and lightbox operations are controlled by defining the onVisaCheckoutReady function that includes event handlers for initialization and purchase events. The function includes the following event handlers:

Event Handler	Description
`V.init`	*(Required)* Event handler for initialization. Specify values for initialization in this handler. Since 2.0
`V.on`	*(Required)* Event handler for Visa Checkout purchase events. The event handler specifies actions to take on the following Visa Checkout events: • `payment.success` • `payment.cancel` • `payment.error` • `pre-payment.user-data-prefill` (since 5.5) Since 2.0

Defining V.init Event Handler

Use the `V.init` event handler to specify a JSON object that contains initialization values for the Visa Checkout JavaScript library. Specify values for the following properties:

Property	Description
`apikey`	*(Required)* The API key created with the Visa Checkout account. Use both a live key and a sandbox key, which are different from each other. **Format:** Alphanumeric; maximum 49 characters **Example:** `apikey=...` Since 2.0
`encryptionKey`	*(Required)* Visa Checkout encrypts data in the consumer information payload using the shared secret associated with this encryption key. **Format:** Alphanumeric; maximum 49 characters **Example:** `encryptionKey=...` Since 6.3
`referenceCallID`	*(Optional)* Visa Checkout transaction ID. The `referenceCallID` can be used with the Preselected Checkout Feature. **Format:** Alphanumeric; maximum 48 characters. **Example:** `"referenceCallID":"..."`

`externalProfileId`	*(Optional)* Profile ID, which is created externally by a merchant or partner, which Visa Checkout uses to populate settings, such as accepted card brands and shipping regions. The properties set in this profile override properties in the merchant's current profile. **Format:** Alphanumeric; maximum 50 characters
`externalClientId`	Not required for merchants. For partners, it is the unique ID associated with a partner's client, such as the ID of a merchant onboarded by the partner. Typically, the external client ID is assigned by a partner; however, Visa Checkout assigns a value if one is not specified. **Note** Some integration strategies, such as that for a hosted order solution, may require specific values to be set; contact your Visa Checkout representative for more information. **Format:** Alphabetic, numeric, hyphens (–), and underscores (_), e.g. spaces are not allowed; maximum 100 characters.
`settings`	*(Optional)* One or more name-value pairs, each of which specifies a configuration attribute. **Format:** A `settings` structure. Since 2.0
`paymentRequest`	*(Optional)* One or more name-value pairs, each of which specifies a payment request attribute. **Format:** A `paymentRequest` structure.

Related Content

Merchant Example

Partner Hosted Merchant Example

Payment Request Properties

Settings Properties

Merchant Example

```
<head>
...
  <script type="text/javascript">
    function onVisaCheckoutReady(){
    V.init({ apikey: "merchantApikey",... });
    V.on("payment.success", function(payment){ ... });
    V.on("payment.cancel", function(payment){ ... });
    V.on("payment.error", function(payment, error){ ... });
```

```
    }
  </script>
...
</head>
```

Related Content

Defining V.init Event Handler (Parent Topic)

Partner Hosted Merchant Example

```
<head>
...
  <script type="text/javascript">
    function onVisaCheckoutReady(){
    V.init({ apikey: "partnerApikey",
             externalClientId: "partnerIDforMerchant"
             externalProfileId: "partnerProfileIDforMerchant" });
    V.on("payment.success", function(payment){ ... });
    V.on("payment.cancel", function(payment){ ... });
    V.on("payment.error", function(payment, error){ ... });
    }
  </script>
...
</head>
```

Related Content

Defining V.init Event Handler (Parent Topic)

Payment Request Properties

Property	Description
`merchantRequestId`	*(Optional)* Merchant's ID associated with the request. Visa Checkout stores this value for your use as a convenience. **Format:** Alphanumeric; maximum 100 characters Since 2.0
`currencyCode`	*(Required)* The currency with which to process the transaction. **Format:** It is one of the following ISO 4217 standard alpha-3 code values: • `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)

Property	Description
	• COP - Colombian Peso (Since 2.7) • EUR - Euro (Since 4.3) • HKD - Hong Kong Dollar (Since 2.7) • INR - Indian rupee (Since 4.6) • KWD - Kuwaiti Dinar (Since 5.1) • 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) • QAR - Qatari Riyal (Since 5.1) • SAR - Saudi Riyal (Since 5.1) • SGD - Singapore Dollar (Since 2.7) • ZAR - Rand (Since 2.7) • AED - UAE Dirham (Since 2.7) • UAH - Ukranian Hryvnia (Since 5.1) • GBP - UK Pound Sterling (Since 4.3) • USD - US Dollar Currency codes must be uppercase. **Example:** `"currencyCode" : "USD"` Since 2.0
`subtotal`	*(Required)* 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
`shippingHandling`	*(Optional)* Total of shipping and handling charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"shippingHandling" : "3.00"` Since 2.0
`tax`	*(Optional)* Total tax-related charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits

Property	Description
	Example: `"tax" : "1.00"` Since 2.0
`discount`	*(Optional)* Total of discounts related to the payment. If provided, it is a positive value representing the amount to be deducted from the total. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"discount" : "2.50"` Since 2.0
`giftWrap`	*(Optional)* Total gift-wrapping charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"giftWrap" : "1.99"` Since 2.0
`misc`	*(Optional)* Total uncategorized charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"misc" : "1.00"` Since 2.0
`total`	*(Optional)* 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
`orderId`	*(Optional)* Merchant's order ID associated with the payment. **Format:** Alphanumeric; maximum 100 characters Since 2.0
`description`	*(Optional)* Description associated with the payment. **Format:** Alphabetic characters, digits, spaces ( ), periods (.), underscores (_), and hyphens (–); maximum 100 characters

Property	Description
	Since 2.0
promoCode	*(Optional)* Promotion codes associated with the payment. **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 **Example:** `"promoCode": "17.15"` Since 2.0
customData	*(Optional)* Merchant-supplied data, as name-value pairs. **Format:** Alphanumeric; maximum 1024 characters **Example:** `customData: { "nvPair": [ { "name": "Name1", "value": "value1" }, { "name": "Name2", "value": "value2" } ] ...` Since 2.0

Related Content

Defining V.init Event Handler (Parent Topic)

Payment Request Configuration Example

Payment Request Configuration Example

You specify the payment request for which the consumer is being asked to agree:

```
V.init({
 ...
 paymentRequest: {
  merchantRequestId: "Merchant defined request ID",
  currencyCode: "USD",
  subtotal: "10.00",
  shippingHandling: "2.00",
  tax: "2.00",
  discount: "1.00",
  giftWrap: "2.00",
  misc: "1.00",
  total: "16.00",
  description: "...corp Product",
  orderId: "Merchant defined order ID",
```

```
    promoCode: "Merchant defined promo code",
    customData: {
     "nvPair": [
      { "name": "customName1", "value": "customValue1" },
      { "name": "customName2", "value": "customValue2" }
     ]
   ...
);
```

Related Content

Payment Request Properties (Parent Topic)

Settings Properties

Property	Description
`locale`	*(Optional)* Override value for the locale, which controls how text displays in the Visa Checkout checkout button and lightbox. **Format:** It is one of the following values: • `es_AR` - Argentina, Spanish (Since 3.5; `en_AR` 2.9-3.4) • `en_AU` - Australia, English • `pt_BR` - Brazil, Portuguese (Since 3.5; `en_BR` 2.9-3.4) • `en_CA` - Canada, English • `fr_CA` - Canada, French • `en_CN` - China, English (Since 2.9) • `zh_CN` - China, Simplified Chinese (Since 3.5) **Note** For display purposes only; input is in English. • `es_CL` - Chile, Spanish (Since 3.5; `en_CL` 2.9-3.4) • `es_CO` - Colombia, Spanish (Since 3.5; `en_CO` 2.9-3.4) • `fr_FR` - France, French (Since 4.3) • `zh_HK` - Hong Kong, Chinese (Since 3.5) **Note** For display purposes only; input is in English. • `en_HK` - Hong Kong, English (Since 2.9) • `en_IN` - India, English (Since 4.6) • `en_IE` - Ireland, English (Since 4.3) • `en_KW` - Kuwait, English (Since 5.1) • `en_MY` - Malaysia, English (Since 2.9) • `es_MX` - Mexico, Spanish (Since 3.5; `en_MX` 2.9-3.4) • `es_PE` - Peru, Spanish (Since 3.5; `en_PE` 2.9-3.4) • `en_NZ` - New Zealand English (Since 2.9) • `pl_PL` - Poland, Polish (Since 4.3) • `en_QA` - Qatar, English (Since 5.1) • `en_SA` - Saudi Arabia, English (Since 5.1)

Property	Description
	• en_SG - Singapore, English • en_ZA - South Africa, English (Since 2.9) • es_ES - Spain, Spanish (Since 4.3) • en_UA - Ukraine, English (Since 5.1) • uk_UA - Ukraine, Ukranian (Since 5.1) **Note** For display purposes only; input is in English. • en_AE - United Arab Emirates, English (Since 2.9) • en_GB - United Kingdom, English (Since 4.3) • en_US - United States, English The value of the locale attribute must be compatible with the value of the country attribute. Since 2.0
countryCode	*(Optional)* Override value for the country code, which controls how text displays in the Visa Checkout checkout button and lightbox. By default, Visa Checkout determines the country from the consumer's IP address. Do not use the countryCode attribute unless explicit control over the display is required. **Format:** One of the following ISO-3166-1 alpha-2 standard codes: • AR - Argentina (Since 2.7) • AU - Australia • BR - Brazil (Since 2.7) • CA - Canada • CL - Chile (Since 2.9) • CN - China (Since 2.9) • CO - Colombia (Since 2.9) • FR - France (Since 4.3) • HK - Hong Kong (Since 2.9) • IN - India (Since 4.6) • IE - Ireland (Since 4.3) • KW - Kuwait (Since 5.1) • MY - Malaysia (Since 2.9) • MX - Mexico (Since 2.9) • NZ - New Zealand (Since 2.9)

Property	Description
	• PE - Peru (Since 2.9) • PL - Poland (Since 4.3) • QA - Qatar (Since 5.1) • SA - Saudi Arabia (Since 5.1) • SG - Singapore • ZA - South Africa (Since 2.9) • ES - Spain (Since 4.3) • UA - Ukraine (Since 5.1) • AE - United Arab Emirates (Since 2.9) • GB - United Kingdom (Since 4.3) • US - United States The value of the country attribute must be compatible with the value of the locale attribute. Since 2.0
displayName	*(Optional)* The merchant's name as it appears on the **Review** panel of the lightbox; typically, it is the name of your company. **Format:** Maximum 100 characters, either alphabetic, numeric, spaces, or the following characters: ! @ # $ % ^ & * – ' ? and period (.) Since 2.0
websiteUrl	*(Optional)* Complete URL to your website. **Format:** Valid URL, beginning with HTTP; maximum 256 characters Since 2.0
customerSupportUrl	*(Optional)* Your complete customer service or support URL. **Format:** Valid URL, beginning with HTTP; maximum 256 characters Since 2.0
shipping	*(Optional)* Shipping properties associated with the lightbox **Format:** shipping Since 2.0
review	*(Optional)* Review properties associated with the lightbox **Format:** review

Property	Description
	Since 2.0
`payment`	*(Optional)* Payment method properties associated with the lightbox **Format:** `payment` Since 2.0
`threeDSSetup`	*(Optional)* Verified by Visa setup properties **Format:** `threeDSSetup` Since 2.8
`dataLevel`	*(Optional)* The level of consumer and payment information that the `payment.success` event response should include. If you request information, permission to receive full information must be configured in Visa Checkout; otherwise, you will always receive only summary information, regardless of the data level you specify. When onboarded by a partner, the `enablePANAccess` field of the onboarding API determines the default value for `dataLevel`. If `enablePANAccess` is `true` when the merchant is onboarded, the default `dataLevel` is `true`; otherwise, the default `dataLevel` is `false`. For information about the `enablePANAccess` field in the onboarding API, see the Client API Reference, Partner Edition. **Format:** It is one of the following values: • `SUMMARY` - Summary information • `FULL` - Full information, which is only available if you are configured to receive it • `NONE` - Consumer and payment information is not returned in the `payment.success` event response, in which case the Get Payment Data API must be used to obtain the information. Since 2.5. Since 2.0

Property	Description
`currencyFormat`	*(Optional)* A string that specifies the display format for a currency amount associated with the **Pay** button in the lightbox. If not set here, the default format displays the amount as `xxx 999,999,999.99`, where `xxx` is the ISO 4217 standard alpha-3 currency code for the currency being used, suppressing leading zeros (`0`) and truncating additional precision in the display; for example, `USD 1,000.00`. The actual value being displayed remains unchanged. **Note** Incorrectly formatted `currencyFormat` values result in the amount being displayed in the default format. **Format:** String that contains the currency display format. **Example:** `"currencyFormat" : "currencyCodeSymbol ###,###,###.##"` Since 4.2
`enableUserDataPrefill`	*(Optional)* Whether the user data prefill event handler is active for this transaction. You must be enabled by Visa Checkout to use the prefill feature; contact Visa Checkout for more information. **Format:** It is one of the following Boolean values: • `true` - Active; the consumer's lightbox can be prefilled • `false` - Inactive; the consumer's lightbox cannot be prefilled (default) **Example**: `"enableUserDataPrefill":true` Since 5.5

Related Content

Defining V.init Event Handler (Parent Topic)

Lightbox Panel Configuration Example

Shipping Properties

Review Properties

Payment Properties

Verified by Visa Setup Properties

Lightbox Panel Configuration Example

You can customize the appearance of lightbox panels, including the language in which text appears, whether the confirmation button is **Continue** or **Pay**, and various messages and ornaments:

```
V.init({
 ...
 settings: {
  locale: "en_US",
  countryCode: "US",
  displayName: "...Corp",
  websiteUrl: "www....Corp.com",
  customerSupportUrl: "www....Corp.support.com",
  ...
  dataLevel: "FULL"
 ...
);
```

Related Content

Settings Properties (Parent Topic)

Shipping Properties

Property	Description
`acceptedRegions`	*(Optional)* Override value for shipping region country codes in the merchant's external default profiles, which limits selection of eligible addresses in the consumer's account. If not set in a profile or overridden here, shipping addresses for all listed countries are allowed. **Format:** One of the following ISO-3166-1 alpha-2 standard codes: • `AR` - Argentina (Since 2.7) • `AU` - Australia • `BR` - Brazil (Since 2.7) • `CA` - Canada • `CL` - Chile (Since 2.9) • `CN` - China (Since 2.9) • `CO` - Colombia (Since 2.9) • `FR` - France (Since 4.3) • `HK` - Hong Kong (Since 2.9) • `IN` - India (Since 4.6) • `IE` - Ireland (Since 4.3) • `KW` - Kuwait (Since 5.1) • `MY` - Malaysia (Since 2.9) • `MX` - Mexico (Since 2.9) • `NZ` - New Zealand (Since 2.9) • `PE` - Peru (Since 2.9) • `PL` - Poland (Since 4.3) • `QA` - Qatar (Since 5.1) • `SA` - Saudi Arabia (Since 5.1) • `SG` - Singapore • `ZA` - South Africa (Since 2.9) • `ES` - Spain (Since 4.3) • `UA` - Ukraine (Since 5.1) • `AE` - United Arab Emirates (Since 2.9) • `GB` - United Kingdom (Since 4.3) • `US` - United States Since 2.0

Property	Description
`collectShipping`	*(Optional)* Whether to obtain a shipping address from the consumer. **Format:** One of the following values: • `true` - Obtain shipping address (default) • `false` - Shipping address is not required Since 2.0

Related Content

Settings Properties (Parent Topic)

Shipping Options Configuration Example

Shipping Options Configuration Example

You can specify whether the consumer can set the shipping address (`collectShipping`) and the regions to which you ship:

```
V.init({
 ...
 settings: {
 ...
  shipping: {
   acceptedRegions: ["US", "CA"],
   collectShipping: "true"
  },
 ...
);
```

Related Content

Shipping Properties (Parent Topic)

Review Properties

Property	Description
`message`	*(Optional)* Your message to display on the **Review** page. You are responsible for translating the message. **Format:** Maximum 100 characters, either alphabetic, numeric, spaces, or the following characters: `! @ # $ % ^ & * – ' ?` and period (`.`). **Note** If you exceed the maximum number of characters, the default review message is displayed. Since 2.0
`buttonAction`	*(Optional)* The button label in the Visa Checkout lightbox. **Format:** One of the following values: • `Continue` - Display **Continue** on the lightbox button (default) • `Pay` - Display **Pay** on the lightbox button **Note** A valid value for `total` must be specified when using **Pay** on the button; otherwise **Continue** will be displayed. Since 2.0

Related Content

Settings Properties (Parent Topic)

Review Options Configuration Example

Review Options Configuration Example

You can specify a message to display in the Visa Checkout lightbox and control the button text:

```
V.init({
  ...
  settings: {
  ...
   review: {
     message: "Review message to display in lightbox",
     buttonAction: "Pay"
   },
  ...
);
```

Related Content

Review Properties (Parent Topic)

Payment Properties

Property	Description
`cardBrands`	*(Optional)* Card brands that are accepted. The `VISA` brand must be included if `acceptCanadianVisaDebit` is `true`. If not set in a profile or overridden here, all listed card brands are accepted. **Format:** Array containing one or more of the following brands: • `VISA` • `MASTERCARD` • `AMEX` • `DISCOVER` • `ELECTRON` (Brazil only; since 3.9) • `ELO` (Brazil only; since 3.9) Since 2.0
`acceptCanadianVisaDebit`	*(Optional)* Override of whether a Canadian merchant accepts Visa Canada debit cards; ignored for non-Canadian merchants. Visa must be specified as an allowable card brand. **Format:** One of the following values: • `true` - Visa Canada debit cards accepted • `false` - Visa Canada debit cards not accepted **Example:** `acceptCanadianVisaDebit : "true"` Since 2.0
`billingCountries`	*(Optional)* Override value for billing country codes in the merchant's external or default profiles, which limits selection of eligible cards in the consumer's account. If not set in a profile or overridden here, payments from all listed billing countries are accepted. **Format:** One of the following ISO-3166-1 alpha-2 standard codes: • `AR` - Argentina (Since 2.7) • `AU` - Australia • `BR` - Brazil (Since 2.7) • `CA` - Canada • `CL` - Chile (Since 2.9) • `CN` - China (Since 2.9)

Property	Description
	• `CO` - Colombia (Since 2.9) • `FR` - France (Since 4.3) • `HK` - Hong Kong (Since 2.9) • `IN` - India (Since 4.6) • `IE` - Ireland (Since 4.3) • `KW` - Kuwait (Since 5.1) • `MY` - Malaysia (Since 2.9) • `MX` - Mexico (Since 2.9) • `NZ` - New Zealand (Since 2.9) • `PE` - Peru (Since 2.9) • `PL` - Poland (Since 4.3) • `QA` - Qatar (Since 5.1) • `SA` - Saudi Arabia (Since 5.1) • `SG` - Singapore • `ZA` - South Africa (Since 2.9) • `ES` - Spain (Since 4.3) • `UA` - Ukraine (Since 5.1) • `AE` - United Arab Emirates (Since 2.9) • `GB` - United Kingdom (Since 4.3) • `US` - United States **Example:** `billingCountries: ["US","CA","AU"]` Since 2.0

Related Content

Settings Properties (Parent Topic)

Payment Options Configuration Example

Payment Options Configuration Example

You can limit the kind of cards you accept:

```
V.init({
 ...
 settings: {
  ...
  payment: {
   cardBrands: [
    "VISA",
    "MASTERCARD"],
   acceptCanadianVisaDebit : "true",
   billingCountries:["US","CA","AU"]
```

```
    },
    ...
   },
  ...
);
```

Related Content

Payment Properties (Parent Topic)

Verified by Visa Setup Properties

Property	Description
`threeDSActive`	*(Optional)* Whether Verified by Visa (VbV) is active for this transaction. If Verified by Visa is configured, you can use `threeDSActive` to deactivate it for the transaction; otherwise, VbV will be active if it has been configured. **Format:**The following value: • `false` - Do not use Verified by Visa for this transaction. Since 2.7
`threeDSSuppress Challenge`	*(Optional)* Whether a Verified by Visa (VbV) consumer authentication prompt is suppressed for this transaction. If `true`, VbV authentication is performed only when it is possible to do so without the consumer prompt. **Format:**It is one of the following values: • `true` - Do not display a consumer prompt. • `false` - Allow a consumer prompt. Since 3.2

Related Content

Settings Properties (Parent Topic)

Deactivate Verified by Visa for a Transaction Example

Suppress Verified by Visa Consumer Prompt Example

Deactivate Verified by Visa for a Transaction Example

```
V.init({
 ...
 settings: {
  ...
  threeDSSetup: {
   threeDSActive : "false"
  },
  ...
```

```
    },
   ...
);
```

Related Content

Verified by Visa Setup Properties (Parent Topic)

Suppress Verified by Visa Consumer Prompt Example

```
V.init({
 ...
 settings: {
  ...
  threeDSSetup: {
   threeDSSuppressChallenge : "true"
  },
  ...
 },
...
);
```

Related Content

Verified by Visa Setup Properties (Parent Topic)

Response to Payment Success Events

The response associated with the `payment.success` event returns the following information:

Property	Description
`callid`	Visa Checkout transaction ID associated with a payment request. By default, the `callid` does not expire. You can request an expiration for a specified period, in days; however, it should be greater than merchant session timeout. **Format:** Alphanumeric; maximum 48 characters **Example:** `"callid":"..."` Since 2.0
`responseStatus`	Response status. **Format:** Response status structure Since 2.0
`encKey`	Encrypted key to be used to decrypt `encPaymentData`. You use your shared secret to decrypt this key. **Format:** Alphanumeric; maximum 128 characters **Example:** `"encKey":"..."`

Property	Description
	Since 2.0
`encPaymentData`	Encrypted consumer and payment data that can be used to process the transaction. You decrypt this by first unwrapping the `encKey` value, then using that unwrapped key to decrypt this value. **Note** For an example decrypted consumer information payload. **Format:** Alphanumeric; maximum 1024 characters **Example:** `"encPaymentData":"..."` Since 2.0
`partialShippingAddress`	Partial shipping address of the consumer. **Format:** Partial shipping address structure Since 2.0
`paymentMethodType`	Type of payment instrument. Present only if you are enabled to receive tokens from Visa Checkout. **Format:** It is one of the following values: • `PAN` — Payload does not contain a `tokenInfo` structure. • `TOKEN` — Payload contains a `tokenInfo` structure. Since 5.4
`vInitRequest`	Ignore this structure.

Related Content

Response Status

Partial Shipping Address

Response Status

Property	Description
`status`	HTTPS response status. **Format:** Numeric Since 2.0
`code`	Internal subcode.

Property	Description
	Format: Numeric Since 2.0
`severity`	Severity of the error. **Format:** It is one of the following values: • `ERROR` • `WARNING` Since 2.0
`message`	Description of the error. **Format:** Alphanumeric Since 2.0

Related Content

Response to Payment Success Events (Parent Topic)

Partial Shipping Address

Property	Description
`countryCode`	Country code of the country where the purchase should be shipped, such as `US`; useful for calculating shipping costs. **Format:** Alphanumeric; maximum 2 characters Since 2.0
`postalCode`	Postal code of the location where the purchase should be shipped, if available; useful for calculating shipping costs. **Format:** Alphanumeric; maximum 128 characters Since 2.0

Related Content

Response to Payment Success Events (Parent Topic)

Response to Payment Cancelled Events

Property	Description
`callid`	Visa Checkout transaction ID that identifies the cancelled payment request. **Format:** Alphanumeric; maximum 48 characters **Example:** `"callid":"..."` Since 2.0

Response to Error Events

Property	Description
`status`	HTTPS response status. **Format:** Numeric Since 2.0
`code`	Internal subcode. **Format:** Numeric Since 2.0
`severity`	Severity of the error. **Format:** It is one of the following values: • `ERROR` • `WARNING` Since 2.0
`message`	Description of the error. **Format:** Alphanumeric Since 2.0

Related Content

Example: Error Event Response

Example: Error Event Response

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

```
}
}
```

Related Content

Response to Error Events (Parent Topic)

User Data Prefill Event Handler

You can prefill a consumer's first name, last name, phone number, and email address in the lightbox from your data for new Visa Checkout consumers. After being enabled by Visa Checkout, you can set the `enableUserDataPrefill` parameter in `V.init` to `true` and provide a `V.on` event handler to provide the consumer information; the event occurs when the consumer accepts the prefill. Consumers are prompted to accept the prefill when they are new, or when they are not recognized as existing Visa Checkout consumers, which can happen when the Visa Checkout cookie is not present in the consumer's browser.

Important

By enabling the prefill consumer information feature, merchants confirm that they have provided all applicable disclosures and/or have obtained all applicable consent from each consumer regarding any intended disclosures and uses of any consumer information to be provided to Visa Checkout. The MSA (Merchant Service Agreement) needs to include language that addresses any support for this feature. Contact your Visa Checkout representative for more information.

From your event handler, you can specify data for any of the following fields; fields that are not specified are not prefilled:

Field	Description
`userFirstName`	(*Optional*) Consumer's given (first) name. **Format:** Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), . (period), and – (hyphen); maximum 24 characters
`userLastName`	(*Optional*) Consumer's surname (last name). **Format:** Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), . (period), and – (hyphen); maximum 24 characters
`userEmail`	(*Required*) Valid email address for the consumer making the payment. **Format:** Alphanumeric, valid email address; maximum 256 characters
`userPhone`	(*Optional*) Valid mobile phone number for the consumer making the payment; for future use. **Format:** Numeric or hyphens, parentheses, period, or plus sign, valid for the country; maximum 30 characters

Your event handler returns the fields that you want to prefill:

```
V.on('pre-payment.user-data-prefill',function(){
  return {
    userFirstName: 'Jill',
    userLastName: 'Consumer',
    userEmail: '...',
    userPhone: '...'
  }
});
```

If the data is not readily available, you can return a promise, which Visa Checkout accepts if the lightbox experience is not degraded by waiting. For example, if you need to query your backend server to obtain the data you want to prefill, you can return a promise and then fetch the data, as shown in the following example:

```
V.on('pre-payment.user-data-prefill',function(){
  return new Promise(function(resolve, reject) {
  // api call to get data and resolve it
  fetch('https:your_endpoint')
    .then(function(data) {
      resolve( {
        userFirstName: data.your_source_for_userFirstName,
        userLastName: data.your_source_for_userLastName,
        userEmail: data.your_source_for_userEmail,
        userPhone: data.your_source_for_userPhone
      })
    })
    .catch(function(error){
      reject(error);
    });
  });
});
```

Complete Visa Checkout Web Page HTML Example

You initialize the Visa Checkout library in the `V.init` event handler of your `onVisaCheckoutReady` function with properties that identify the merchant implementing the button, button characteristics and settings related to the behavior of the lightbox, and payment request properties. You specify how to respond to events related to the lightbox closing and the payment request in `V.on` event handlers.

Note

You must provide your API key when initializing the Visa Checkout JavaScript library.

The following example shows an HTML web page that loads the Visa Checkout library, defines handlers for initialization and payment events, and creates a Visa Checkout button:

```
<html>
 <head>
  <script type="text/javascript">
   function onVisaCheckoutReady() {
     V.init({
     apikey: "...",
     encryptionKey: "...",

     settings: {
      locale: "en_US",
      countryCode: "US",
      displayName: "...Corp",
```

```
      websiteUrl: "www....Corp.com",
      customerSupportUrl: "www....Corp.support.com",
      enableUserDataPrefill:true,
      shipping: {
       acceptedRegions: ["US", "CA"],
       collectShipping: "true"
      },
      payment: {
       cardBrands: [
        "VISA",
        "MASTERCARD"],
       acceptCanadianVisaDebit: "true",
       billingCountries:["US","CA"]
      },
      review: {
       message: "Merchant defined message",
       buttonAction: "Continue"
      },
     dataLevel: "SUMMARY"
     },
     paymentRequest: {
      merchantRequestId: "Merchant defined request ID",
      currencyCode: "USD",
      subtotal: "10.00",
      shippingHandling: "2.00",
      tax: "2.00",
      discount: "1.00",
      giftWrap: "2.00",
      misc: "1.00",
      total: "16.00",
      description: "...corp Product",
      orderId: "Merchant defined order ID",
      promoCode: "Merchant defined promo code",
      customData: {
       "nvPair": [
       { "name": "customName1", "value": "customValue1" },
       { "name": "customName2", "value": "customValue2" }
      ]
     }
    }
   }
  );
  V.on("payment.success", function(payment){document.write(JSON.stringify(payment));
  V.on("payment.cancel", function (payment) { ... });
  V.on("payment.error", function (payment, error) { ... });
  V.on("pre-payment.user-data-prefill", function(){ ... });
}
 </script>
</head>
<body>
 <img alt="Visa Checkout" class="v-button" role="button"
 src="https://sandbox.secure.checkout.visa.com/wallet-services-web/xo/button.png?
  cardBrands=VISA,MASTERCARD,DISCOVER,AMEX"/>
 <script type="text/javascript"
  src="https://sandbox-assets.secure.checkout.visa.com/
       checkout-widget/resources/js/integration/v1/sdk.js">
 </script>
</body>
</html>
```

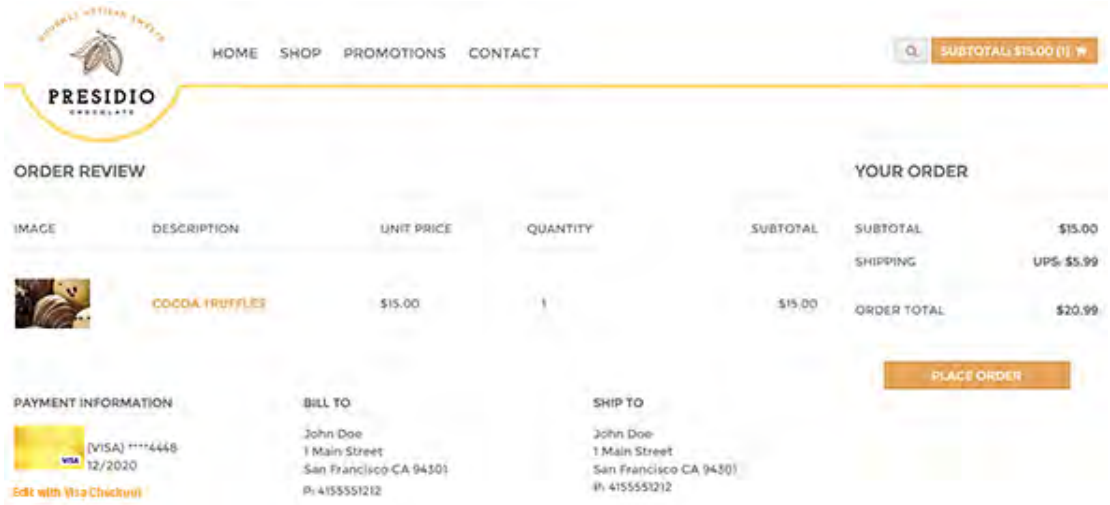
Preselected Checkout Feature

The preselected checkout feature allows you to set the initial values for the consumer's card, shipping address, and billing address, based on those used in a previous payment request, identified by a call ID. This feature enables you to offer the consumer a way to change the card or address before confirming a payment. If you retain call IDs, you can also use this feature in other ways; for example, to establish a card on file for a future payment request.

To preselect the consumer's card, shipping address, and billing address for a payment:

1. Specify the call ID of a previous request in the `referenceCallID` attribute of the `V.init` structure.
2. Invoke your `onVisaCheckoutReady` function to set these values and invoke the Visa Checkout lightbox.

Consider the following example, which displays payment information on an order confirmation page and contains an **Edit with Visa Checkout** link, which enables the consumer to change the payment information:



The following example shows the source, which sets `referenceCallID` in the `onVisaCheckoutReady` function and invokes the function through the SDK when the consumer clicks the **Edit with Visa Checkout** link:

```
<head>
 <script type="text/javascript">
  function onVisaCheckoutReady() {
   V.init({
     apikey: "…",
     referenceCallID: "554449457741154050”,
     …
   });
   V.on("payment.success", function(payment)
      {document.write(JSON.stringify(payment)); });
   …
   });
  }
 </script>
</head>
<body>
 <a class="v-button" href="#">Edit with Visa Checkout</a>
 <script type="text/javascript"
```

```
 src="https://sandbox-assets.secure.checkout.visa.com/
      checkout-widget/resources/js/integration/v1/sdk.js">
 </script>
</body>
```

Note

Each `payment.success` event generates a new call ID.

Mobile App Support 4

Summary of Mobile App Options

Visa Checkout provides SDKs that integrate Visa Checkout into native iOS and Android apps. It also provides a hybrid SDK, which enables you to integrate your web app with biometric authentication provided by iOS and Android devices.

Visa Checkout provides the following SDKs, which are available from the Visa Developer Center at https://developer.visa.com:

- *Visa Checkout SDK for iOS* for native iOS apps
- *Visa Checkout SDK for Android* for native Android apps
- *Visa Checkout Hybrid Mobile SDK* for web apps using the Visa Checkout plugin for biometric authentication

Documentation for each Visa Checkout mobile SDK is in the SDK.

You can use these SDKs or you can create a web view in your hybrid app to display Visa Checkout assets.

Mobile App Examples

Related Content

iOS Web View Hybrid App

Android Web View Hybrid App

Optimizing the Checkout Flow for Mobile Browsers

iOS Web View Hybrid App

You can associate your website with an iOS web view. Your code must be self-contained, meaning that it should not interact with your iOS hybrid app after it is loaded; all Visa Checkout integration should be performed in the website.

The following example shows how to load the web view and associate it with your site:

```
import UIKit
import WebKit

class ViewController: UIViewController {
  var webView: WKWebView!

  override func viewDidLoad() {
    super.viewDidLoad()
    webView = WKWebView(frame: view.frame)
    view.addSubview(webView)
    let url = URL(string: "https://my_site.my_company.com/...")
    let urlRequest = URLRequest(url: url!)
    webView.load(urlRequest)

  }
}
```

If you cannot use a web kit view, you can load the web view as follows:

```
import UIKit

class ViewController: UIViewController {
  @IBOutlet weak var webView: UIWebView!

  override func viewDidLoad() {
    super.viewDidLoad()
    let url = URL(string: "https://my_site.my_company.com/...")
    let urlRequest = URLRequest(url: url!)
    webView.loadRequest(urlRequest)

  }
}
```

Related Content

Mobile App Examples (Parent Topic)

Android Web View Hybrid App

You can associate your website with an Android web view. Your code must be self-contained, meaning that it should not interact with your Android hybrid app after it is loaded; all Visa Checkout integration should be performed in the website.

The following example shows how to create the web view and associate it with your site:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.webkit.WebView;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    WebView webView = (WebView)findViewById(R.id.web_view);
    webView.loadUrl("https://my_site.my_company.com/...");
  }
}
```

Related Content

Mobile App Examples (Parent Topic)

Optimizing the Checkout Flow for Mobile Browsers

Visa Checkout is optimized for mobile browsers even if your checkout flow is not. Support is provided for both iOS and Android devices. In order to allow for a mobile optimized Visa Checkout experience, add the following `<meta>` tag to your HTML `<head>` block:

```
<html>
<head>
...
<meta name="viewport"
content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
```

```
user-scalable=no" />
...
</head>
```

Related Content

Mobile App Examples (Parent Topic)

Enabling Third-Party Cookies for Hybrid Apps

To enable the display of a Visa Checkout button's associated card art and the Remember Me feature when the Visa Checkout lightbox is invoked from a hybrid app, you must allow the acceptance of third-party cookies. To accept third-party cookies, you must add code to your hybrid app, as follows:

Related Content

Accepting Cookies in iOS Hybrid Apps

Accepting Cookies in Android Hybrid Apps

Accepting Cookies in iOS Hybrid Apps

```
[NSHTTPCookieStorage sharedHTTPCookieStorage].cookieAcceptPolicy
 = NSHTTPCookieAcceptPolicyAlways;
```

Related Content

Enabling Third-Party Cookies for Hybrid Apps (Parent Topic)

Accepting Cookies in Android Hybrid Apps

```
if(Build.VERSION.SDK_INT >= 21) {
CookieManager.getInstance().setAcceptThirdPartyCookies(myWebView, true);
}
```

Related Content

Enabling Third-Party Cookies for Hybrid Apps (Parent Topic)

Consumer Information 5

About Consumer Information

Consumer information is returned in JSON format. You must not rely on the position of structures or fields in the payload as being fixed. Neither should you rely upon the existence of fields, such as fields that are contextually inapplicable because they may not be returned. You should consider using standard libraries to parse JSON objects. Consumer information is available, either encrypted in a payload or as summary information from Get Payment Data.

Consumer Information

Field	Description
`externalClientId`	The partner's relationship ID for a merchant, or a Visa Checkout-supplied ID if a partner did not specify a value for `externalClientId` when onboarding the merchant. **Format:** String. Alphabetic, numeric, hyphens (-), and underscores (_), e.g. spaces are not allowed; maximum 100 characters **Example:** `"externalClientId":"123456"`
`paymentRequest`	Payment request information from the Visa Checkout library initialization. **Format:** A `paymentRequest` structure.
`userData`	Consumer payment information. **Format:** A `userData` structure.
`creationTimeStamp`	Payment creation timestamp. **Format:** UNIX Epoch timestamp, in milliseconds. **Example:** `"creationTimeStamp":"1397847423768"`.
`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:** A `paymentInstrument` structure.

`shippingAddress`	Shipping address information. **Format:** An `Address` structure.
`riskData`	Risk information. **Format:** A `riskData` structure.
`threeDS`	Verified by Visa (3-D Secure) information. (See 3–D Secure Authentication Data Fields) **Format:** A `threeDS` structure.
`visaCheckoutGuest`	Guest Checkout. **Do not use.** **Format:** It is the following value: • `false` **Example:** `"visaCheckoutGuest":"false"`
`newUser`	Whenever a consumer-enrolls for the first time, the value is `true`; otherwise, this field is not returned. It is one of the following values: • `true` - returns value for first-time user • `false` - returns value for any user who is not a first-time user **Format:** A `newUser` structure.
`walletInfo`	Wallet information. **Format:** A `walletInfo` structure.
`partialShippingAddress`	Partial shipping address. **Format:** A `partialShippingAddress` structure.
`paymentMethodType`	Type of payment instrument. Present only if you are enabled to receive tokens from Visa Checkout. **Format:** It is one of the following values: • `PAN` — Payload does not contain a `tokenInfo` structure. • `TOKEN` — Payload contains a `tokenInfo` structure.

Related Content

Payment Request

User Data

Payment Instrument Properties

Address

Risk Properties

3–D Secure Authentication Data Fields

Wallet Info

Partial Shipping Address

Payment Request

Field	Description
`merchantRequestId`	Merchant's ID associated with the request. Visa Checkout stores this value for your use as a convenience. **Format:** Alphanumeric; maximum 100 characters.
`currencyCode`	The currency with which to process the transaction. **Format:** Currency codes must be uppercase; ISO 4217 standard alpha-3 code values. **Example:** `"currencyCode":"USD"`
`subtotal`	Subtotal of the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits. **Example:** `"subtotal":"9.00"`
`shippingHandling`	Total of shipping and handling charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits. **Example:** `"shippingHandling" : "3.00"`
`tax`	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"`
`discount`	Total of discounts related to the payment. If provided, it is a positive value representing the amount to be deducted from the total. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits. **Example:** `"discount" : 2.50"`
`giftWrap`	Total gift-wrapping charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits.

	Example: `"giftWrap" : "1.99"`
`misc`	Total uncategorized charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits. **Example:** `"misc":"1.00"`
`total`	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"`
`description`	The description of the payment. **Format:** Alphabetic characters, digits, spaces ( ), periods (.), underscores (_), and hyphens (—); maximum 100 characters.
`orderId`	Merchant's order ID associated with the payment. **Format:** Alphanumeric; maximum 100 characters.
`promoCode`	*(Optional)* Promotion codes associated with the payment. **Format:** String. Alphabetic, numeric, characters, 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. **Example:** `"promoCode": "17.15"`
`customData`	Merchant-supplied data, as name-value pairs. **Format:** Alphanumeric; maximum 1024 characters.

Related Content

Consumer Information (Parent Topic)

User Data

Field	Description
`userFirstName`	Consumer's given (first) name.

	Format: Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), . (period), and – (hyphen); maximum 256 characters. **Example:** `"userFirstName" : "Joe"`
`userLastName`	Consumer's surname (last name). **Format:** Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), . (period), and – (hyphen); maximum 256 characters. **Example:** `"userLastName" : "Tester"`
`userFullName`	Concatenation of consumer's first and last names. **Format:** Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), . (period), and – (hyphen); maximum 256 characters. **Example:** `"userFullName" : "Joe Tester"`
`userName`	User name. **Format:** Alphanumeric, valid email address or mobile phone number; maximum 256 characters. **Example:** `"userName" : "testuser@mycompany.com"`
`encUserId`	Encoded user ID. **Format:** Alphanumeric; maximum 100 characters. **Example:** `"encUserId" : "..."`
`userEmail`	Valid email address for the consumer making the payment. **Format:** Alphanumeric, valid email address; maximum 256 characters. **Example:** `"userEmail" : "testuser@mycompany.com"`

`userMobile`	Valid mobile phone number for the consumer making the payment; for future use. **Format:** Numeric or hyphens, parentheses, period, or plus sign, valid for the country; maximum 30 characters. **Example:** `"userMobile" : "..."`
`userPersonalId`	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. **Format:** Numeric; maximum 11 digits. **Example:** `"userPersonalId": "56453472856"`

Related Content

Consumer Information (Parent Topic)

Payment Instrument Properties

Field	Description
`id`	Unique internal ID associated with the payment instrument. **Format:** Alphanumeric. **Example:** `"id" : "..."` Since 2.0
`lastFourDigits`	Last 4 digits of the payment instrument. **Format:** Numeric; maximum 4 digits. **Example:** `"lastFourDigits" : "4448"` Since 3.0
`tokenInfo`	Token information; only available for token-enabled payment instruments. **Format:** A `tokenInfo` structure. Since 3.4
`cryptogramInfo`	Cryptogram information associated with the token; only available for tokenized payment instruments. **Format:** A `cryptogramInfo` structure. Since 3.4

`binSixDigits`	First 6 digits of account number-based payment instrument. **Note** Not present for tokenized payment instruments. **Format:** Numeric; maximum 6 digits **Example:** `"binSixDigits" : "444444"` Since 3.0
`accountNumber`	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. **Note** Not present for tokenized payment instruments. **Format:** Numeric; maximum 19 digits. **Example:** `"accountNumber" : "..."` Since 2.0
`paymentType`	Kind of payment instrument. **Format:** A `paymentType` structure. Since 2.0
`paymentAccountReference`	An ID that is assigned by Visa to provide an association with a PAN and any tokens that are related to a PAN. **Format:** Alphanumeric; maximum 29 digits. **Example:** `"paymentAccountReference" : "V0010013818052688164702673046"`
`billingAddress`	Billing address associated with the payment instrument. **Format:** A `billingAddress` structure. Since 2.0
`verificationStatus`	Visa Checkout verification status for the payment instrument. **Format:** It is one of the following values: • `VERIFIED` • `NOT_VERIFIED`

	• CONSUMER_OVERRIDE" **Example:** "verificationStatus" : "VERIFIED" Since 2.0
expired	Whether the card has expired. **Format:** It is one of the following values: • true - Expired • false - Not expired **Example:** "expired" : "false" Since 2.0
cardArts	Card art information. **Format:** A cardArts structure. Since 2.0
issuerBid	Issuer BID. **Format:** Alphanumeric; maximum 100 characters. **Example:** "issuerBid" : "14" Since 2.0
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** 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. **Format:** An `expirationDate` structure. Since 2.0
`riskData`	Risk information related to the transaction, if available. **Format:** A `riskData` structure. Since 2.0

Related Content

Consumer Information (Parent Topic)

Token Info Properties

Cryptogram Info Properties

Payment Type Properties

Card Art

Expiration Date

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`	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. **Format:** Numeric; maximum 16 digits **Example:** `"token" : "..."` Since 3.4
`tokenRange`	First 9 digits of the token. **Format:**Numeric; maximum 9 digits

Field	Description
	Example: `"tokenRange" : "123444444"` Since 3.4
`last4`	Last 4 digits of the token. **Format:**Numeric; maximum 4 digits **Example:** `"last4" : "1234"` Since 3.4
`expirationDate`	Token's expiration date. **Format:** `expirationDate` Since 3.4

Related Content

Payment Instrument Properties (Parent Topic)

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`	Current cryptogram associated with the token **Note** A token authentication verification value (TAVV) cryptogram must adhere to the VisaNet requirements for Field 126.9, which is 20 bytes long. The cryptogram must be decoded into a 20–byte binary value before submitting it to VisaNet. Visa Checkout creates a new cryptogram via every payload request with the Get Payment Data API. **Format:** Alphanumeric **Example:** `"cryptogramInfo" : "..."` Since 3.4
`eci`	An e-commerce indicator (ECI) represents a value to indicate the authentication results of a consumer's credit card payment on a secure channel **Important** If you are using Verified by Visa (3–D Secure) with tokens, you must provide the `eciRaw` value in the `threeDS` structure to your processor, not this value. **Format:** It returns the following 2–digit value:

Field	Description
	• `07` **Example:** `"eci" : "07"` Since 3.4
`tokenCryptoType`	Type of token **Format:** 4–letter code **Example:** `"tokenCryptoType" : "TAVV"` Since 5.5
`expirationTimestamp`	Date and time when the cryptogram expires, in milliseconds **Format:** ISO 8601 standard in the form of *yyyy-mm-ddThh:mm:ss:mmmZ* **Example:** `"expirationTimestamp" : "2017-07-22T07:07:59.000Z"` Since 5.5

Related Content

Payment Instrument Properties (Parent Topic)

Payment Type Properties

Field	Description
`cardBrand`	Brand of payment instrument. **Format:** It is one of the following values: • `VISA` • `MASTERCARD` • `AMEX` • `DISCOVER` • `ELECTRON` (Brazil only; since 3.9) • `ELO` (Brazil only; since 3.9) **Example:** `"cardBrand" : "VISA"` Since 2.0
`cardType`	Kind of card. **Format:** It is one of the following values: • `CREDIT` • `DEBIT` • `CHARGE` • `PREPAID` • `DEFERRED DEBIT` • `NONE` **Example:** `"cardType" : "CREDIT"` Since 2.0

Related Content

Payment Instrument Properties (Parent Topic)

Card Art

Zero or more Card Art structures

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

Related Content

Payment Instrument Properties (Parent Topic)

Expiration Date

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

Related Content

Payment Instrument Properties (Parent Topic)

Address

The following information describes address payload fields, irrespective of country. For country-specific formats, see the *Visa Checkout Address Format by Country* document.

Property	Description
`id`	Address ID **Note** Only available for shipping addresses **Format:** Alphanumeric; maximum 36 characters **Example:** `"id":"..."` Since 2.0
`verificationStatus`	Visa Checkout verification status of the address. **Note** Only available for shipping addresses **Format:** It is one of the following values: • `VERIFIED` • `NOT_VERIFIED` • `CONSUMER_OVERRIDE` **Example:** `"verificationStatus" : "VERIFIED"` Since 2.0
`personName`	Addressee's complete name. **Format:** Alphanumeric; maximum 256 characters **Example:** `"personName" : "Test User"` Since 2.0
`firstName`	Addressee's first name. **Format:** Alphanumeric; maximum 256 characters **Example:** `"firstName" : "Test"` Since 2.9
`lastName`	Addressee's last name. **Format:** Alphanumeric; maximum 256 characters **Example:** `"lastName" : "User"`

Property	Description
	Since 2.9
`line1`	First line of the address. **Format:** Alphanumeric, which depends on the country; maximum 140 characters. For country-specific information, see *Visa Checkout Address Formats by Country*. **Example:** `"line1" : "1 Main Street"` Since 2.0
`streetNumber`	Street number in the first line of the address, if it exists. It is used only for addresses in Mexico and Brazil. **Format:** Alphanumeric; maximum 10 characters **Example:** `"streetNumber": "1738"` Since 3.9
`streetName`	Street name in the first line of the address, if it exists. It is used only for addresses in Mexico and Brazil. **Format:** Alphanumeric, which depends on the country; maximum 140 characters. For country-specific information, see *Visa Checkout Address Formats by Country*. **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 addresses in Mexico and Brazil. **Format:** Alphanumeric; maximum 10 characters **Example:** `"additionalLocation": "Apt 4"` Since 3.9
`neighborhood`	Neighborhood or colony. It is used only for addresses in Mexico. **Format:** Alphanumeric; maximum 69 characters **Example:** `"neighborhood" : "..."` Since 4.4
`county`	Delegation or municipality. It is used only for addresses in Mexico. **Format:** Alphanumeric; maximum 69 characters

Property	Description
	Example: `"county" : "..."` Since 4.4
`pointOfReference`	Point of reference. It is used only for addresses in Mexico. **Format:** Alphanumeric; maximum 140 characters **Example:** `"pointOfReference" : "..."` Since 4.4
`line2`	Second line of the address, if it exists. **Format:** Alphanumeric, which depends on the country; maximum 140 characters. For country-specific information, see *Visa Checkout Address Formats by Country*. **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, which depends on the country; maximum 140 characters. For country-specific information, see *Visa Checkout Address Formats by Country*. 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 in US, CA, or AU, or the county name for IE. 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
`postalCode`	Postal code associated with the address, such as a zip code. **Format:** Depends on `stateProvinceCode`, maximum 100 characters:

Property	Description
	• 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 Other postal codes must be valid for their respective countries, if a code exists. **Example:** "postalCode" : "94301" Since 2.0
countryCode	Country code associated with the address. **Format:** One of the following ISO-3166-1 alpha-2 standard codes: • AR - Argentina (Since 2.7) • AU - Australia • BR - Brazil (Since 2.7) • CA - Canada • CL - Chile (Since 2.9) • CN - China (Since 2.9) • CO - Colombia (Since 2.9) • FR - France (Since 4.3) • HK - Hong Kong (Since 2.9) • IN - India (Since 4.6) • IE - Ireland (Since 4.3) • KW - Kuwait (Since 5.1) • MY - Malaysia (Since 2.9) • MX - Mexico (Since 2.9) • NZ - New Zealand (Since 2.9) • PE - Peru (Since 2.9) • PL - Poland (Since 4.3) • QA - Qatar (Since 5.1) • SA - Saudi Arabia (Since 5.1) • SG - Singapore • ZA - South Africa (Since 2.9) • ES - Spain (Since 4.3) • UA - Ukraine (Since 5.1) • AE - United Arab Emirates (Since 2.9)

Property	Description
	• `GB` - United Kingdom (Since 4.3) • `US` - United States **Example:** `"countryCode" : "US"`
`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: • `true` • `false` **Example:** `"default" : false` Since 2.5

Related Content

Consumer Information (Parent Topic)

Risk Properties

Property	Description
`advice`	**Not currently available.** Risk advice for use with your fraud model. **Important** 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. **Format:** It is one of the following values: • `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 **Example:** `"advice" : "LOW"` Since 2.0
`score`	Risk score; 0 indicates unavailable. **Not currently available; always 0.** The higher the score, the higher the perceived risk. **Important** 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. **Format:** Numeric; whole value between 0 and 99, inclusive **Example:** `"score" : "0"` Since 2.0
`avsResponseCode`	Address verification system response code. **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. **Format:** Alphanumeric **Example:** `"avsResponseCode" : "V"` Since 2.0

Property	Description
`cvvResponseCode`	Card validation verification system response code. **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. **Format:** Alphanumeric **Example:** `"cvvResponseCode" : "M"` Since 2.0
`ageOfAccount`	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. **Format:** Numeric **Example:** `"ageOfAccount" : 300` Since 2.8
`firstUseAuthenticated`	Whether a payment instrument has previously been authenticated during first use in Visa Checkout. This field only appears for consumers in designated countries. **Format:** It is one of the following Boolean values: • `true` - The payment instrument has previously been authenticated during first use in Visa Checkout. • `false` - The payment instrument has not been authenticated during first use in Visa Checkout. **Example:** `"firstUseAuthenticated" : false` Since 5.1

Related Content

Consumer Information (Parent Topic)

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), Mastercard (SecureCode) card, or American Express (SafeKey) by consumers for configured merchants in designated countries, 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 authorization message to your processor. Consult with your processor about the fields and values to include in the authorization message.

Field	Description
`eciRaw`	A brand-specific e-commerce indicator (ECI); for example, a successful authentication of a Visa card returns `05` for `eciRaw`. **Important** If you are using Verified by Visa (3–D Secure) with tokens, you must provide the `eciRaw` value to your processor, not the `eci` value in the the `cryptogramInfo` structure. **Format:** One of the following values: • `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 and American Express (American Express on consumer first use in Europe only) • `06` - Visa and American Express (American Express on consumer first use in Europe only) • `07` - Visa and American Express (American Express on consumer first use in Europe only) **Example:** `"eciRaw" : "05"` Since 2.7
`cavv`	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. **Format:** Alphanumeric; maximum 48 characters **Example:** `"cavv" : "..."` Since 2.7
`veresEnrolled`	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.

	Format: It is one of the following values: • `Y` - Enrolled • `N` - Not enrolled • `B` - Authentication bypassed • `U` - Authentication unavailable **Example:** `"veresEnrolled" : "U"` Since 2.7
`veresTimestamp`	VERes response timestamp in Coordinated Universal Time (UTC), also known as Zulu time. **Format:** ISO 8601 standard in the form of yyyy-mm-ddThh:mm:ss:mmmZ. **Example:** `"veresTimestamp":"2014-12-29T18:34:45.489Z"` Since 2.7
`paresStatus`	Whether the transaction was successfully authenticated. **Format:** It is one of the following values: • `Y` - Authenticated • `N` - Not authenticated • `U` - Authentication could not be completed • `A` - Successful attempts transaction **Example:** `"paresStatus" : "U"` Since 2.7
`paresTimestamp`	PARes response timestamp in Coordinated Universal Time (UTC), also known as Zulu time. **Format:** ISO 8601 standard in the form of yyyy-mm-ddThh:mm:ss:mmmZ. **Example:** `"paresTimestamp":"2014-12-29T18:37:07.413Z"` Since 2.7

	Whether the PARes has been validated successfully. **Format:** It is one of the following values: • `Y` - Indicates that the signature of the PARes has been validated successfully. • `N` - Indicates that the PARes could not be validated. **Example:** `"signatureVerification" : "Y"` Since 3.5
`xId`	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. **Format:** Alphanumeric; maximum 40 characters. **Example:** `"xId":"..."` Since 2.7

Related Content

Consumer Information (Parent Topic)

Wallet Info

Field	Description
`walletName`	The pay wallet indicator. **Format:** It is one of the following values: • `SAMSUNG_PAY` • `ANDRIOD_PAY` • `VISA_CHECKOUT` **Example:** `"walletName":"VISA_CHECKOUT"` Since 5.6

Related Content

Consumer Information (Parent Topic)

Partial Shipping Address

Field	Description
countryCode	Country code of the country where an item should be shipped, such as US; useful for calculating shipping costs. **Format:** One of the following ISO-3166-1 alpha-2 standard codes: • AR - Argentina (Since 2.7) • AU - Australia • BR - Brazil (Since 2.7) • CA - Canada • CL - Chile (Since 2.9) • CN - China (Since 2.9) • CO - Colombia (Since 2.9) • FR - France (Since 4.3) • HK - Hong Kong (Since 2.9) • IN - India (Since 4.6) • IE - Ireland (Since 4.3) • KW - Kuwait (Since 5.1) • MY - Malaysia (Since 2.9) • MX - Mexico (Since 2.9) • NZ - New Zealand (Since 2.9) • PE - Peru (Since 2.9) • PL - Poland (Since 4.3) • QA - Qatar (Since 5.1) • SA - Saudi Arabia (Since 5.1) • SG - Singapore • ZA - South Africa (Since 2.9) • ES - Spain (Since 4.3) • UA - Ukraine (Since 5.1) • AE - United Arab Emirates (Since 2.9) • GB - United Kingdom (Since 4.3) • US - United States Since 2.0
postalCode	Postal code of the location where an item should be shipped, if available; useful for calculating shipping costs.

Field	Description
	Format: Depends on `countryCode`, maximum 7 characters: • `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 Since 2.0

Related Content

Consumer Information (Parent Topic)

Get Payment Data 6

Get Payment Data Summary

Call the `payment/data` resource path at the appropriate endpoint to obtain information about the Visa Checkout transaction.

This API returns the same consumer information that is returned by the `payment.success` event in the frontend; it is provided as a convenience if you want your backend software to process the data.

Notes

1. The difference between full and summary information is whether or not the full account number is returned; if it is not returned, the information is summary. Full information is encrypted because it contains the account number. Summary information is not encrypted.
2. You can call `payment/data` multiple times for a `callId`, up to the time the call ID expires if an expiration is set; the data may change between calls.
3. By default, the `callId` does not expire. You can request an expiration for a specified period, in days; however, it should be greater than the merchant's session timeout and, if storing the call ID for future use, it should be greater than the anticipated last-use date. Do not request an expiration if you accept tokenized payments.
4. A partner can specify the merchant ID, which was assigned as the external client ID (`externalClientId`), and the `callId` to retrieve payment data on behalf of their merchant. The partner's API key is used to specify the caller.
5. Merchants who save the `CallId` on file and invoke the GetPaymentData API receive the latest PAN updates that the VAU has completed.

Get Payment Data Request

Path and Endpoints

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

Complete endpoint:

Sandbox:

```
https://sandbox.api.visa.com/wallet-services-web/payment/data/{callId}
  ?apikey=key,encryptionKey=encryptionKey,dataLevel=level,externalClientId=id
```

Live:

```
https://api.com/wallet-services-web/payment/data/{callId}
  ?apikey=key,encryptionKey=encryptionKey,dataLevel=level,externalClientId=id
```

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

Method

`GET`

Required Headers

Header	Description
`x-pay-token`	A token identifying the transaction and its contents. **Format:** Alphanumeric; maximum 100 characters in the form of `xv2:`*`UTC_Timestamp`*`:`*`HMAC-SHA256_hash`*, where • *UTC_Timestamp* is a UNIX Epoch timestamp • *HMAC-SHA256_hash* is an HMAC-SHA256 hash using the shared secret associated with your API key and the following unseparated items: 1. Timestamp from the transaction; exactly the same as *UNIX_UTC_Timestamp* 2. Resource path (API name) 3. This HTTPS request's query string, if it exists **Note** To create the query string, concatenate all query string components (names and values) as UTF-8 characters, which are URL-encoded per RFC 3986. Hex characters must be uppercase. Multiple parameters must be sorted using lexicographic byte ordering and separated from each other by an ampersand (`&`) character (ASCII code 38). Parameter names are separated from their values by the = character (ASCII character 61), which must be present even if the value is empty. "Unreserved" characters specified in Section 2.3 of *RFC 3986* , including dash -, dot `.`, underscore _, and tilde ~ should not be URL-encoded. 4. Complete request body, when a request body exists **Example:** `x-pay-token: xv2:1440199445:`*`HMAC-SHA256_hash result`* Since 6.3

`Accept`	Acceptable response format.
	
	Format: Must include the following value:
	
	`application/json`
	
	`application/xml`
	
	Example: `Accept: application/json`
	
	Example: `Accept: application/xml`
	
	Since 2.0

Header	Description

`Content-Type`	Format of the content.
	Format: Must include the following value:
	`application/json`
	`application/xml`
	Example: `Content-Type: application/json`
	Example: `Content-Type: application/xml`
	Since 2.0

Query Parameters

Parameter	Description
`apikey`	*(Required)* Public API key, which is different than the shared secret. **Format:** Alphanumeric; maximum 49 characters **Example:** `apikey=...` Since 2.0
`encryptionKey`	Visa Checkout encrypts data in the consumer information payload using the shared secret associated with this encryption key. **Format:** Alphanumeric; maximum 49 characters **Example:** `encryptionKey=...` Since 6.3

`dataLevel`	*(Optional)* Whether the response should include summary information or full information. This value overrides the value in your external profile, if it is set. When onboarded by a partner, the `enablePANAccess` field of the onboarding API determines the default value for `dataLevel`. If `enablePANAccess` is `true` when the merchant is onboarded, the default `dataLevel` is `true`; otherwise, the default `dataLevel` is `false`. For information about the `enablePANAccess` field in the onboarding API, see the Client API Reference, Partner Edition. **Format:** It is one of the following values: • `SUMMARY` - Summary information • `FULL` - Full information, which is only available if you are configured to receive it Since 2.0
`externalClientId`	*(Optional)* Partner's ID for the merchant receiving the payment. A partner must set this value when obtaining a consumer information payload on behalf of their merchant; merchants should not specify this value. **Format:** Alphabetic, numeric, hyphens (–), and underscores (_), e.g., spaces are not allowed; maximum 100 characters Since 4.4

Get Payment Data Response

The response properties depend on whether full or summary information was requested and, if full information was requested, whether your Visa Checkout permissions allow full information to be included in the response. If full information is permitted and requested, you receive a response that includes encrypted payment information, which you must decrypt to obtain the full payment information. If you request summary information, you receive summary payment information, which is not encrypted.

Note

Unencrypted summary payment information is exactly the same as decrypted full payment information, except that the `accountNumber` property is not returned.

Related Content

Full Payment Information Before Decryption

Full Payment Information Before Decryption

Field	Description
`encKey`	Encrypted key to be used to decrypt `encPaymentData`. You use your shared secret to decrypt this key. **Format:** Alphanumeric; maximum 128 characters **Example:** `"encKey":"..."` Since 2.0
`encPaymentData`	Encrypted consumer data that can be used to process the transaction. You decrypt this by first unwrapping the `encKey` value, then using that unwrapped key to decrypt this value. **Format:** Alphanumeric **Example:** `"encPaymentData":"..."` Since 2.0
`partialShippingAddress`	Partial shipping address; useful for shipping calculations. Since 2.0
`paymentMethodType`	Type of payment instrument. Present only if you are enabled to receive tokens from Visa Checkout. **Format:** It is one of the following values: • `PAN` — Payload does not contain a `tokenInfo` structure. • `TOKEN` — Payload contains a `tokenInfo` structure. Since 5.4

Related Content

Get Payment Data Response (Parent Topic)

Partial Shipping Address

Partial Shipping Address

Field	Description
countryCode	Country code of the country where an item should be shipped, such as US; useful for calculating shipping costs. **Format:** One of the following ISO-3166-1 alpha-2 standard codes: • AR - Argentina (Since 2.7) • AU - Australia • BR - Brazil (Since 2.7) • CA - Canada • CL - Chile (Since 2.9) • CN - China (Since 2.9) • CO - Colombia (Since 2.9) • FR - France (Since 4.3) • HK - Hong Kong (Since 2.9) • IN - India (Since 4.6) • IE - Ireland (Since 4.3) • KW - Kuwait (Since 5.1) • MY - Malaysia (Since 2.9) • MX - Mexico (Since 2.9) • NZ - New Zealand (Since 2.9) • PE - Peru (Since 2.9) • PL - Poland (Since 4.3) • QA - Qatar (Since 5.1) • SA - Saudi Arabia (Since 5.1) • SG - Singapore • ZA - South Africa (Since 2.9) • ES - Spain (Since 4.3) • UA - Ukraine (Since 5.1) • AE - United Arab Emirates (Since 2.9) • GB - United Kingdom (Since 4.3) • US - United States Since 2.0
postalCode	Postal code of the location where an item should be shipped, if available; useful for calculating shipping costs.

Field	Description
	Format: Depends on `countryCode`, maximum 7 characters: • `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 Since 2.0

Related Content

Full Payment Information Before Decryption (Parent Topic)

Get Payment Data Error Response

Response Status

Property	Description
`status`	HTTPS response status. **Format:** Numeric Since 2.0
`code`	Internal subcode. **Format:** Numeric Since 2.0
`severity`	Severity of the error. **Format:** It is one of the following values: • `ERROR` • `WARNING` Since 2.0
`message`	Description of the error. **Format:** Alphanumeric Since 2.0

Example

```
...
"responseStatus" : {
 "status" : 200,
 "code" : 1017,
```

```
 "severity" : "WARNING",
 "message" : "Requested data access level [FULL] is more detailed
  than merchant profile level [SUMMARY], assuming the latter."
},
...
```

Get Payment Data Errors

Status	Code	Description
400	1015	Invalid request data; a required field is either missing or invalid
400	1035	Shipping region is not accepted by the merchant.
401	1033	The requested data access level (`dataLevel`) is invalid.
403	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	1079	`externalClientId` request parameter does not match `externalClientId` in payload. Verify the request `externalClientId` and `callId`.
403	None	`x-pay-token` header missing or invalid, or API key is missing or invalid
403	1058	Customer's account is locked.
403	1059	Customer's account is closed.
403	1073	Further operations on the card are not allowed.
404	1010	API key or call ID not found, or data referenced by the API key or call ID is invalid or not available.
404	1065	Expired Call ID
404	1076	The token-enabled card is not found; it may be deleted
409	1074	Invalid token request

Note

Other errors are internal.

Get Payment Data Examples

Get Summary Payment Data Success Example—Merchant

JSON Request Including Headers

```
GET
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/data/03e...
?apikey=...&dataLevel=SUMMARY
Accept: application/json
X-PAY-TOKEN: X:1397847496:...
```

Related Content

JSON Response—Account Number-Based Summary Payment Instrument

JSON Response—Token-Enabled Summary Payment Instrument

JSON Response—Account Number-Based Summary Payment Instrument


```
{
 "paymentRequest" : {
  "customData" : {
   "nvPair" : [
    {
     "name" : "customName1",
     "value" : "customValue1"
    },
    {
     "name" : "customName2",
     "value" : "customValue2"
    }
   ]
  },
  "merchantRequestId" : "898",
  "currencyCode" : "USD",
  "subtotal" : "80",
  "shippingHandling" : "5",
  "tax" : "5",
  "discount" : "5",
  "giftWrap" : "10",
  "misc" : "5",
  "total" : "100",
  "orderId" : "testorderID",
  "promoCode" : "testCampaignId"
 },
 "userData" : {
  "userFirstName" : "Checkout",
  "userLastName" : "Team",
  "userFullName" : "Checkout Team",
  "userName" : "7084410557",
  "userEmail" : "Checkout094192@gmail.com",
  "encUserId" : "..."
 },
 "creationTimeStamp" : 1397847423768,
 "paymentInstrument" : {
  "id" : "...",
```

```
 "lastFourDigits" : "1221",
 "binSixDigits" : "371449",
 "paymentType" : {
  "cardBrand" : "VISA",
  "cardType" : "CREDIT"
 },
 "billingAddress" : {
  "personName" : "Joe Tester",
  "firstName" : "Joe",
  "lastName" : "Tester",
  "line1" : "...",
  "city" : "Foster City",
  "stateProvinceCode" : "CA",
  "postalCode" : "94404",
  "countryCode" : "US"
 },
 "verificationStatus" : "VERIFIED",
 "expired" : false,
 "cardArts" : {
  "cardArt" : [
   {
    "baseImageFileName" : "https://....png",
    "height" : 50,
    "width" : 77
   }
  ]
 },
 "issuerBid" : "null",
 "nickName" : "...",
 "nameOnCard" : "Joe Tester",
 "cardFirstName" : "Joe",
 "cardLastName" : "Tester",
 "expirationDate" : {
  "month" : "10",
  "year" : "2015"
 }
},
"shippingAddress" : {
 "id" : "...=",
 "verificationStatus" : "NOT_VERIFIED",
 "personName" : "Joe Tester",
 "firstName" : "Joe",
 "lastName" : "Tester",
 "line1" : "...",
 "city" : "Foster City",
 "stateProvinceCode" : "CA",
 "postalCode" : "94404",
 "countryCode" : "US"
},
"riskData" : {
 "advice" : "Unavailable",
 "score" : 0,
 "avsResponseCode" : "Y",
 "cvvResponseCode" : "M"
},
"visaCheckoutGuest": false,
"walletInfo": {
  "walletName": "VISA_CHECKOUT"
},
```

```
 "newUser": true,
 "partialShippingAddress" : {
  "countryCode" : "US",
  "postalCode" : "94404"
 },
  "paymentMethodType": "PAN"
}
```

Related Content

JSON Request Including Headers (Parent Topic)

JSON Response—Token-Enabled Summary Payment Instrument

```
{
 "paymentRequest" : {
  "customData" : {
   "nvPair" : [
    {
     "name" : "customName1",
     "value" : "customValue1"
    },
    {
     "name" : "customName2",
     "value" : "customValue2"
    }
   ]
  },
  "merchantRequestId" : "898",
  "currencyCode" : "USD",
  "subtotal" : "80",
  "shippingHandling" : "5",
  "tax" : "5",
  "discount" : "5",
  "giftWrap" : "10",
  "misc" : "5",
  "total" : "100",
  "orderId" : "testorderID",
  "promoCode" : "testCampaignId"
 },
 "userData" : {
  "userFirstName" : "Checkout",
  "userLastName" : "Team",
  "userFullName" : "Checkout Team",
  "userName" : "7084410557",
  "userEmail" : "Checkout094192@gmail.com",
  "encUserId" : "..."
 },
 "creationTimeStamp" : 1441931603973,
 "paymentInstrument" : {
  "id" : "...",
  "lastFourDigits" : "4684",
  "tokenInfo" : {
    "tokenRange" : "405954033",
    "last4" : "0206",
    "expirationDate" : {
      "month" : "10",
      "year" : "2018"
```

```
    }
  },
  "paymentType" : {
   "cardBrand" : "VISA",
   "cardType" : "CREDIT"
  },
  "paymentAccountReference": "V001...",
  "billingAddress" : {
   "personName" : "Joe Tester",
  "firstName" : "Joe",
  "lastName" : "Tester",
   "line1" : "...",
   "city" : "Foster City",
   "stateProvinceCode" : "CA",
   "postalCode" : "94404",
   "countryCode" : "US"
   "phone" : "6505551212",
   "default": false
  },
  "verificationStatus" : "VERIFIED",
  "expired" : false,
  "cardArts" : {
   "cardArt" : [
    {
     "baseImageFileName" : "https://....png",
     "height" : 50,
     "width" : 77
    }
   ]
  },
  "issuerBid" : "null",
  "nickName" : "...",
  "nameOnCard" : "Joe Tester",
  "cardFirstName" : "Joe",
  "cardLastName" : "Tester",
},
"shippingAddress" : {
 "id" : "...=",
 "verificationStatus" : "NOT_VERIFIED",
 "personName" : "Joe Tester",
 "firstName" : "Joe",
 "lastName" : "Tester",
 "line1" : "...",
 "city" : "Foster City",
 "stateProvinceCode" : "CA",
 "postalCode" : "94404",
 "countryCode" : "US"
 "phone" : "6505551212",
 "default": false
},
"riskData" : {
 "advice" : "Unavailable",
 "score" : 0,
 "avsResponseCode" : "Y",
 "cvvResponseCode" : "M"
},
"visaCheckoutGuest": false,
"walletInfo": {
```

```
   "walletName": "VISA_CHECKOUT"
 },
 "partialShippingAddress" : {
  "countryCode" : "US",
  "postalCode" : "94404"
 },
  "paymentMethodType": "TOKEN"
}
```

Related Content

JSON Request Including Headers (Parent Topic)

XML Request Including Headers

```
GET
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/data/03e...
?apikey=...&dataLevel=SUMMARY
Accept: application/xml
X-PAY-TOKEN: X:1397847496:...
```

Related Content

XML Response—Account Number-Based Summary Payment Instrument

XML Response—Token-Enabled Summary Payment Instrument

XML Response—Account Number-Based Summary Payment Instrument

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:getPaymentDataResponse
...
 <partialShippingAddress>
  <countryCode>US</countryCode>
  <postalCode>94404</postalCode>
 </partialShippingAddress>
 <paymentMethodType>PAN</paymentMethodType>
 <ns2:paymentRequest>
  <customData>
   <nvPair>
    <name>customName1</name>
    <value>customValue1</value>
   </nvPair>
   <nvPair>
    <name>customName2</name>
    <value>customValue2</value>
   </nvPair>
  </customData>
  <merchantRequestId>898</merchantRequestId>
  <currencyCode>USD</currencyCode>
  <subtotal>80</subtotal>
  <shippingHandling>5</shippingHandling>
  <tax>5</tax>
  <discount>5</discount>
  <giftWrap>10</giftWrap>
  <misc>5</misc>
  <total>100</total>
  <orderId>testorderID</orderId>
```

```
 <promoCode>testCampaignId</promoCode>
</ns2:paymentRequest>
<ns2:userData>
 <ns2:userFirstName>Checkout</ns2:userFirstName>
 <ns2:userLastName>Team</ns2:userLastName>
 <ns2:userFullName>Checkout Team</ns2:userFullName>
 <ns2:userName>7084410557</ns2:userName>
 <ns2:userEmail>Checkout094192@gmail.com</ns2:userEmail>
 <ns2:encUserId>...</ns2:encUserId>
</ns2:userData>
<ns2:creationTimeStamp>2014-04-18T18:57:03.768Z</ns2:creationTimeStamp>
<ns2:paymentInstrument>
 <nickName>...</nickName>
 <nameOnCard>...</nameOnCard>
 <cardFirstName>...</cardFirstName>
 <cardLastName>...</cardLastName>
 <expirationDate>
  <month>10</month>
  <year>2015</year>
 </expirationDate>
 <id>...</id>
 <lastFourDigits>1221</lastFourDigits>
 <binSixDigits>371449</binSixDigits>
 <paymentType>
  <cardBrand>VISA</cardBrand>
  <cardType>CREDIT</cardType>
 </paymentType>
 <billingAddress>
 <personName>...</personName>
 <firstName>...</firstName>
 <lastName>...</lastName>
  <line1>...</line1>
  <city>Foster City</city>
  <stateProvinceCode>CA</stateProvinceCode>
  <postalCode>94404</postalCode>
  <countryCode>US</countryCode>
  <phone>6505551212</phone>
 <default>false</default>
 </billingAddress>
 <verificationStatus>VERIFIED</verificationStatus>
 <expired>false</expired>
 <cardArts>
  <cardArt>
   <baseImageFileName>https://....png</baseImageFileName>
   <height>50</height>
   <width>77</width>
  </cardArt>
 </cardArts>
 <issuerBid>null</issuerBid>
</ns2:paymentInstrument>
<ns2:shippingAddress>
 <personName>...</personName>
 <firstName>...</firstName>
 <lastName>...</lastName>
 <line1>...</line1>
 <city>Foster City</city>
 <stateProvinceCode>CA</stateProvinceCode>
 <postalCode>94404</postalCode>
 <countryCode>US</countryCode>
```

```
  <phone>6505551212</phone>
  <default>false</default>
  <id>...</id>
  <verificationStatus>NOT_VERIFIED</verificationStatus>
 </ns2:shippingAddress>
 <ns2:riskData>
  <advice>Unavailable</advice>
  <score>0</score>
  <avsResponseCode>Y</avsResponseCode>
  <cvvResponseCode>M</cvvResponseCode>
 </ns2:riskData>
</ns2:getPaymentDataResponse>
```

Related Content

XML Request Including Headers (Parent Topic)

XML Response—Token-Enabled Summary Payment Instrument

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:getPaymentDataResponse
...
 <partialShippingAddress>
  <countryCode>US</countryCode>
  <postalCode>94404</postalCode>
 </partialShippingAddress>
 <paymentMethodType>TOKEN</paymentMethodType>
 <ns2:paymentRequest>
  <customData>
   <nvPair>
    <name>customName1</name>
    <value>customValue1</value>
   </nvPair>
   <nvPair>
    <name>customName2</name>
    <value>customValue2</value>
   </nvPair>
  </customData>
  <merchantRequestId>898</merchantRequestId>
  <currencyCode>USD</currencyCode>
  <subtotal>80</subtotal>
  <shippingHandling>5</shippingHandling>
  <tax>5</tax>
  <discount>5</discount>
  <giftWrap>10</giftWrap>
  <misc>5</misc>
  <total>100</total>
  <orderId>testorderID</orderId>
  <promoCode>testCampaignId</promoCode>
 </ns2:paymentRequest>
 <ns2:userData>
  <ns2:userFirstName>Checkout</ns2:userFirstName>
  <ns2:userLastName>Team</ns2:userLastName>
  <ns2:userFullName>Checkout Team</ns2:userFullName>
  <ns2:userName>7084410557</ns2:userName>
  <ns2:userEmail>Checkout094192@gmail.com</ns2:userEmail>
  <ns2:encUserId>...</ns2:encUserId>
 </ns2:userData>
```

```
 <ns2:creationTimeStamp>2015-09-11T00:33:23.973Z</ns2:creationTimeStamp>
 <ns2:paymentInstrument>
  <nickName>...</nickName>
  <nameOnCard>...</nameOnCard>
  <cardFirstName>...</cardFirstName>
  <cardLastName>...</cardLastName>
  <id>...</id>
  <lastFourDigits>4684</lastFourDigits>
  <tokenInfo>
   <tokenRange>405954033</tokenRange>
   <last4>0206</last4>
   <expirationDate>
    <month>10</month>
    <year>2018</year>
   </expirationDate>
  </tokenInfo>
  <paymentType>
   <cardBrand>VISA</cardBrand>
   <cardType>CREDIT</cardType>
   <paymentAccountReference>V001...</paymentAccountReference>  </paymentType>
<billingAddress>
  <personName>...</personName>
  <firstName>...</firstName>
  <lastName>...</lastName>
   <line1>...</line1>
   <city>Foster City</city>
   <stateProvinceCode>CA</stateProvinceCode>
   <postalCode>94404</postalCode>
   <countryCode>US</countryCode>
   <phone>6505551212</phone>
    <default>false</default>
</billingAddress>
  <verificationStatus>VERIFIED</verificationStatus>
  <expired>false</expired>
  <cardArts>
   <cardArt>
    <baseImageFileName>https://....png</baseImageFileName>
    <height>50</height>
    <width>77</width>
   </cardArt>
  </cardArts>
  <issuerBid>null</issuerBid>
 </ns2:paymentInstrument>
 <ns2:shippingAddress>
  <personName>...</personName>
  <firstName>...</firstName>
  <lastName>...</lastName>
  <line1>...</line1>
  <city>Foster City</city>
  <stateProvinceCode>CA</stateProvinceCode>
  <postalCode>94404</postalCode>
  <countryCode>US</countryCode>
  <phone>6505551212</phone>
  <default>false</default>
  <id>...</id>
  <verificationStatus>NOT_VERIFIED</verificationStatus>
 </ns2:shippingAddress>
 <ns2:riskData>
  <advice>Unavailable</advice>
```

```
  <score>0</score>
  <avsResponseCode>Y</avsResponseCode>
  <cvvResponseCode>M</cvvResponseCode>
 </ns2:riskData>
</ns2:getPaymentDataResponse>
```

Related Content

XML Request Including Headers (Parent Topic)

Get Summary Payment Data Success Example—Partner

Only use this version if you are acting on behalf of one of your merchants and the merchant's API key was used to initially generate the payload.

For more information, see Visa Checkout JavaScript and Button Reference.

Related Content

JSON Request Including Headers

XML Request Including Headers

JSON Request Including Headers

```
GET
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/data/03e...
?externalClientId=partner_assigned_id_for_merchant
&apikey=partner_api_key
Accept: application/json
X-PAY-TOKEN: X:1397847496:...
```

Related Content

Get Summary Payment Data Success Example—Partner (Parent Topic)

JSON Response—Account Number-Based Summary Payment Instrument

JSON Response—Token-Enabled Summary Payment Instrument

JSON Response—Account Number-Based Summary Payment Instrument

```
{
 "externalClientId": "partner_assigned_id_for_merchant",
 "paymentRequest" : {
  "customData" : {
   "nvPair" : [
    {
     "name" : "customName1",
     "value" : "customValue1"
    },
    {
     "name" : "customName2",
     "value" : "customValue2"
    }
   ]
  },
  "merchantRequestId" : "898",
```

```
 "currencyCode" : "USD",
 "subtotal" : "80",
 "shippingHandling" : "5",
 "tax" : "5",
 "discount" : "5",
 "giftWrap" : "10",
 "misc" : "5",
 "total" : "100",
 "orderId" : "testorderID",
 "promoCode" : "testCampaignId"
},
"userData" : {
 "userFirstName" : "Checkout",
 "userLastName" : "Team",
 "userFullName" : "Checkout Team",
 "userName" : "7084410557",
 "userEmail" : "Checkout094192@gmail.com",
 "encUserId" : "..."
},
"creationTimeStamp" : 1397847423768,
"paymentInstrument" : {
 "id" : "...",
 "lastFourDigits" : "1221",
 "binSixDigits" : "371449",
 "paymentType" : {
  "cardBrand" : "VISA",
  "cardType" : "CREDIT"
 },
 "billingAddress" : {
  "personName" : "Joe Tester",
  "firstName" : "Joe",
  "lastName" : "Tester",
  "line1" : "...",
  "city" : "Foster City",
  "stateProvinceCode" : "CA",
  "postalCode" : "94404",
  "countryCode" : "US"
 },
 "verificationStatus" : "VERIFIED",
 "expired" : false,
 "cardArts" : {
  "cardArt" : [
   {
    "baseImageFileName" : "https://....png",
    "height" : 50,
    "width" : 77
   }
  ]
 },
 "issuerBid" : "null",
 "nickName" : "...",
 "nameOnCard" : "Joe Tester",
 "cardFirstName" : "Joe",
 "cardLastName" : "Tester",
 "expirationDate" : {
  "month" : "10",
  "year" : "2015"
 }
},
```

```
 "shippingAddress" : {
  "id" : "...=",
  "verificationStatus" : "NOT_VERIFIED",
  "personName" : "Joe Tester",
  "firstName" : "Joe",
  "lastName" : "Tester",
  "line1" : "...",
  "city" : "Foster City",
  "stateProvinceCode" : "CA",
  "postalCode" : "94404",
  "countryCode" : "US"
 },
 "riskData" : {
  "advice" : "Unavailable",
  "score" : 0,
  "avsResponseCode" : "Y",
  "cvvResponseCode" : "M"
 },
 "visaCheckoutGuest": false,
 "walletInfo": {
   "walletName": "VISA_CHECKOUT"
 },
 "partialShippingAddress" : {
  "countryCode" : "US",
  "postalCode" : "94404"
 },
  "paymentMethodType": "PAN"
}
```

Related Content

JSON Request Including Headers (Parent Topic)

JSON Response—Token-Enabled Summary Payment Instrument

```
{
 "externalClientId": "partner_assigned_id_for_merchant",
 "paymentRequest" : {
  "customData" : {
   "nvPair" : [
    {
     "name" : "customName1",
     "value" : "customValue1"
    },
    {
     "name" : "customName2",
     "value" : "customValue2"
    }
   ]
  },
  "merchantRequestId" : "898",
  "currencyCode" : "USD",
  "subtotal" : "80",
  "shippingHandling" : "5",
  "tax" : "5",
  "discount" : "5",
  "giftWrap" : "10",
  "misc" : "5",
```

```
 "total" : "100",
 "orderId" : "testorderID",
 "promoCode" : "testCampaignId"
},
"userData" : {
 "userFirstName" : "Checkout",
 "userLastName" : "Team",
 "userFullName" : "Checkout Team",
 "userName" : "7084410557",
 "userEmail" : "Checkout094192@gmail.com",
 "encUserId" : "..."
},
"creationTimeStamp" : 1441931603973,
"paymentInstrument" : {
 "id" : "...",
 "lastFourDigits" : "4684",
 "tokenInfo" : {
   "tokenRange" : "405954033",
   "last4" : "0206",
   "expirationDate" : {
     "month" : "10",
     "year" : "2018"
   }
 },
 "paymentType" : {
  "cardBrand" : "VISA",
  "cardType" : "CREDIT"
 },
 "paymentAccountReference": "V001...",
 "billingAddress" : {
  "personName" : "Joe Tester",
 "firstName" : "Joe",
 "lastName" : "Tester",
  "line1" : "...",
  "city" : "Foster City",
  "stateProvinceCode" : "CA",
  "postalCode" : "94404",
  "countryCode" : "US"
  "phone" : "6505551212",
  "default": false
 },
 "verificationStatus" : "VERIFIED",
 "expired" : false,
 "cardArts" : {
  "cardArt" : [
   {
    "baseImageFileName" : "https://....png",
    "height" : 50,
    "width" : 77
   }
  ]
 },
 "issuerBid" : "null",
 "nickName" : "...",
 "nameOnCard" : "Joe Tester",
 "cardFirstName" : "Joe",
 "cardLastName" : "Tester",
},
"shippingAddress" : {
```

```
  "id" : "...=",
  "verificationStatus" : "NOT_VERIFIED",
  "personName" : "Joe Tester",
  "firstName" : "Joe",
  "lastName" : "Tester",
  "line1" : "...",
  "city" : "Foster City",
  "stateProvinceCode" : "CA",
  "postalCode" : "94404",
  "countryCode" : "US"
  "phone" : "6505551212",
  "default": false
 },
 "riskData" : {
  "advice" : "Unavailable",
  "score" : 0,
  "avsResponseCode" : "Y",
  "cvvResponseCode" : "M"
 },
 "visaCheckoutGuest": false,
 "walletInfo": {
   "walletName": "VISA_CHECKOUT"
 },
 "partialShippingAddress" : {
  "countryCode" : "US",
  "postalCode" : "94404"
 },
  "paymentMethodType": "TOKEN"
}
```

Related Content

JSON Request Including Headers (Parent Topic)

XML Request Including Headers

```
GET
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/data/03e...
?externalClientId=partner_assigned_id_for_merchant
&apikey=partner_api_key
&dataLevel=SUMMARY
Accept: application/xml
X-PAY-TOKEN: X:1397847496:...
```

Related Content

Get Summary Payment Data Success Example—Partner (Parent Topic)

XML Response—Account Number-Based Summary Payment Instrument

XML Response—Token-Enabled Summary Payment Instrument

XML Response—Account Number-Based Summary Payment Instrument

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

```
...
 <partialShippingAddress>
  <countryCode>US</countryCode>
  <postalCode>94404</postalCode>
 </partialShippingAddress>
 <ns2:paymentRequest>
 <paymentMethodType>PAN</paymentMethodType>
  <customData>
   <nvPair>
    <name>customName1</name>
    <value>customValue1</value>
   </nvPair>
   <nvPair>
    <name>customName2</name>
    <value>customValue2</value>
   </nvPair>
  </customData>
  <merchantRequestId>898</merchantRequestId>
  <currencyCode>USD</currencyCode>
  <subtotal>80</subtotal>
  <shippingHandling>5</shippingHandling>
  <tax>5</tax>
  <discount>5</discount>
  <giftWrap>10</giftWrap>
  <misc>5</misc>
  <total>100</total>
  <orderId>testorderID</orderId>
  <promoCode>testCampaignId</promoCode>
 </ns2:paymentRequest>
 <ns2:userData>
  <ns2:userFirstName>Checkout</ns2:userFirstName>
  <ns2:userLastName>Team</ns2:userLastName>
  <ns2:userFullName>Checkout Team</ns2:userFullName>
  <ns2:userName>7084410557</ns2:userName>
  <ns2:userEmail>Checkout094192@gmail.com</ns2:userEmail>
  <ns2:encUserId>...</ns2:encUserId>
 </ns2:userData>
 <ns2:creationTimeStamp>2014-04-18T18:57:03.768Z</ns2:creationTimeStamp>
 <ns2:paymentInstrument>
  <nickName>...</nickName>
  <nameOnCard>...</nameOnCard>
  <cardFirstName>...</cardFirstName>
  <cardLastName>...</cardLastName>
  <expirationDate>
   <month>10</month>
   <year>2015</year>
  </expirationDate>
  <id>...</id>
  <lastFourDigits>1221</lastFourDigits>
  <binSixDigits>371449</binSixDigits>
  <paymentType>
   <cardBrand>VISA</cardBrand>
   <cardType>CREDIT</cardType>
  </paymentType>
  <billingAddress>
  <personName>...</personName>
  <firstName>...</firstName>
  <lastName>...</lastName>
   <line1>...</line1>
```

```
   <city>Foster City</city>
   <stateProvinceCode>CA</stateProvinceCode>
   <postalCode>94404</postalCode>
   <countryCode>US</countryCode>
   <phone>6505551212</phone>
  <default>false</default>
  </billingAddress>
  <verificationStatus>VERIFIED</verificationStatus>
  <expired>false</expired>
  <cardArts>
   <cardArt>
    <baseImageFileName>https://....png</baseImageFileName>
    <height>50</height>
    <width>77</width>
   </cardArt>
  </cardArts>
  <issuerBid>null</issuerBid>
 </ns2:paymentInstrument>
 <ns2:shippingAddress>
  <personName>...</personName>
  <firstName>...</firstName>
  <lastName>...</lastName>
  <line1>...</line1>
  <city>Foster City</city>
  <stateProvinceCode>CA</stateProvinceCode>
  <postalCode>94404</postalCode>
  <countryCode>US</countryCode>
  <phone>6505551212</phone>
  <default>false</default>
  <id>...</id>
  <verificationStatus>NOT_VERIFIED</verificationStatus>
 </ns2:shippingAddress>
 <ns2:riskData>
  <advice>Unavailable</advice>
  <score>0</score>
  <avsResponseCode>Y</avsResponseCode>
  <cvvResponseCode>M</cvvResponseCode>
 </ns2:riskData>
</ns2:getPaymentDataResponse>
```

Related Content

XML Request Including Headers (Parent Topic)

XML Response—Token-Enabled Summary Payment Instrument

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:getPaymentDataResponse
...
 <partialShippingAddress>
  <countryCode>US</countryCode>
  <postalCode>94404</postalCode>
 </partialShippingAddress>
 <paymentMethodType>TOKEN</paymentMethodType>
 <ns2:paymentRequest>
  <customData>
   <nvPair>
    <name>customName1</name>
```

```
    <value>customValue1</value>
   </nvPair>
   <nvPair>
    <name>customName2</name>
    <value>customValue2</value>
   </nvPair>
  </customData>
  <merchantRequestId>898</merchantRequestId>
  <currencyCode>USD</currencyCode>
  <subtotal>80</subtotal>
  <shippingHandling>5</shippingHandling>
  <tax>5</tax>
  <discount>5</discount>
  <giftWrap>10</giftWrap>
  <misc>5</misc>
  <total>100</total>
  <orderId>testorderID</orderId>
  <promoCode>testCampaignId</promoCode>
 </ns2:paymentRequest>
 <ns2:userData>
  <ns2:userFirstName>Checkout</ns2:userFirstName>
  <ns2:userLastName>Team</ns2:userLastName>
  <ns2:userFullName>Checkout Team</ns2:userFullName>
  <ns2:userName>7084410557</ns2:userName>
  <ns2:userEmail>Checkout094192@gmail.com</ns2:userEmail>
  <ns2:encUserId>...</ns2:encUserId>
 </ns2:userData>
 <ns2:creationTimeStamp>2015-09-11T00:33:23.973Z</ns2:creationTimeStamp>
 <ns2:paymentInstrument>
  <nickName>...</nickName>
  <nameOnCard>...</nameOnCard>
  <cardFirstName>...</cardFirstName>
  <cardLastName>...</cardLastName>
  <id>...</id>
  <lastFourDigits>4684</lastFourDigits>
  <tokenInfo>
   <tokenRange>405954033</tokenRange>
   <last4>0206</last4>
   <expirationDate>
    <month>10</month>
    <year>2018</year>
   </expirationDate>
  </tokenInfo>
  <paymentType>
   <cardBrand>VISA</cardBrand>
   <cardType>CREDIT</cardType>
   <paymentAccountReference>V001...</paymentAccountReference>  </paymentType>
<billingAddress>
  <personName>...</personName>
  <firstName>...</firstName>
  <lastName>...</lastName>
   <line1>...</line1>
   <city>Foster City</city>
   <stateProvinceCode>CA</stateProvinceCode>
   <postalCode>94404</postalCode>
   <countryCode>US</countryCode>
   <phone>6505551212</phone>
    <default>false</default>
</billingAddress>
```

```
  <verificationStatus>VERIFIED</verificationStatus>
  <expired>false</expired>
  <cardArts>
   <cardArt>
    <baseImageFileName>https://....png</baseImageFileName>
    <height>50</height>
    <width>77</width>
   </cardArt>
  </cardArts>
  <issuerBid>null</issuerBid>
 </ns2:paymentInstrument>
 <ns2:shippingAddress>
  <personName>...</personName>
  <firstName>...</firstName>
  <lastName>...</lastName>
  <line1>...</line1>
  <city>Foster City</city>
  <stateProvinceCode>CA</stateProvinceCode>
  <postalCode>94404</postalCode>
  <countryCode>US</countryCode>
  <phone>6505551212</phone>
  <default>false</default>
  <id>...</id>
  <verificationStatus>NOT_VERIFIED</verificationStatus>
 </ns2:shippingAddress>
 <ns2:riskData>
  <advice>Unavailable</advice>
  <score>0</score>
  <avsResponseCode>Y</avsResponseCode>
  <cvvResponseCode>M</cvvResponseCode>
 </ns2:riskData>
</ns2:getPaymentDataResponse>
```

Related Content

XML Request Including Headers (Parent Topic)

Get Full Payment Data Success Example

JSON Request Including Headers

```
GET
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/data/03e...
?apikey=...&dataLevel=FULL
Accept: application/json
X-PAY-TOKEN: X:1397847496:...
```

Related Content

JSON Response—Account Number-Based Full Payment Instrument

JSON Response—Token-Enabled Full Payment Instrument

JSON Response—Account Number-Based Full Payment Instrument

```
{
  "partialShippingAddress" : {
```

```
    "countryCode" : "US",
    "postalCode" : "94404"
},
  "encPaymentData" : "...",
  "encKey" : "...",
  "paymentMethodType": "PAN"
}
```

Note

The decrypted payload is the same payload as in the summary, except for the inclusion of `accountNumber` in `paymentInstrument`:

```
"paymentInstrument": {
"id": "...",
"lastFourDigits": "1221",
"binSixDigits": "371449",
"accountNumber": "371449...1221",
    ...
```

Related Content

JSON Request Including Headers (Parent Topic)

JSON Response—Token-Enabled Full Payment Instrument

```
{
  "partialShippingAddress" : {
    "countryCode" : "US",
    "postalCode" : "94404"
},
  "encPaymentData" : "...",
  "encKey" : "..."",
  "paymentMethodType": "TOKEN"
}
```

Note

The decrypted payload is the same payload as in the summary, except for the inclusion of `token` in `tokenInfo` and `cryptogramInfo` in `paymentInstrument`:

```
"paymentInstrument": {
  "id": "...",
  "lastFourDigits": "4684",
  "tokenInfo": {
    "token": "4059540330000206",
    "tokenRange": "405954033",
    "last4": "0206",
    "expirationDate": {
      "month": "10",
      "year": "2018"
    }
  },
  "cryptogramInfo": {
    "cryptogram": "...",
    "eci": "07"
    "tokenCryptoType": "TAVV",
    "expirationTimestamp": "2017-07-22T07:07:59.000Z"
  },
  "paymentAccountReference": "V001...",
...
```

Related Content

JSON Request Including Headers (Parent Topic)

XML Request Including Headers

```
GET
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/data/03e...
?apikey=...&dataLevel=FULL
Accept: application/xml
X-PAY-TOKEN: X:1397847496:...
```

Related Content

XML Response—Account Number-Based Full Payment Instrument

XML Response—Token-Enabled Full Payment Instrument

XML Response—Account Number-Based Full Payment Instrument

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:getPaymentDataResponse
...
 <partialShippingAddress>
  <countryCode>US</countryCode>
  <postalCode>94404</postalCode>
 </partialShippingAddress>
 <encPaymentData>...</encPaymentData>
 <encKey>...</encKey>
 <paymentMethodType>TOKEN</paymentMethodType>
</ns2:getPaymentDataResponse>
```

Note

The decrypted payload is the same payload as in the summary, except for the inclusion of `accountNumber` in `paymentInstrument`:

```
 <ns2:paymentInstrument>
  ...
  <id>...</id>
  <lastFourDigits>4684</lastFourDigits>
  <binSixDigits>371449</binSixDigits>
  <accountNumber>...</accountNumber>
  ...
```

Related Content

XML Request Including Headers (Parent Topic)

XML Response—Token-Enabled Full Payment Instrument

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:getPaymentDataResponse
...
 <partialShippingAddress>
  <countryCode>US</countryCode>
  <postalCode>94404</postalCode>
 </partialShippingAddress>
 <encPaymentData>...</encPaymentData>
```

```
 <encKey>...</encKey>
 <paymentMethodType>TOKEN</paymentMethodType>
</ns2:getPaymentDataResponse>
```

Note

The decrypted payload is the same payload as in the summary, except for the inclusion of `token` in `tokenInfo` and `cryptogramInfo` in `paymentInstrument`:

```
 <ns2:paymentInstrument>
 ...
  <id>...</id>
  <lastFourDigits>4684</lastFourDigits>
  <tokenInfo>
   <token>4059540330000206</token>
   <tokenRange>405954033</tokenRange>
   <last4>0206</last4>
   <expirationDate>
    <month>10</month>
    <year>2018</year>
   </expirationDate>
  </tokenInfo>
  <cryptogramInfo>
   <cryptogram>...</cryptogram>
   <eci>07</eci>
   <tokenCryptoType>TAVV</tokenCryptoType>
   <expirationTimestamp>2017-07-22T07:07:59.000Z</expirationTimestamp>
  </cryptogramInfo>
  <paymentAccountReference>V001...</paymentAccountReference>
  ...
```

Related Content

XML Request Including Headers (Parent Topic)

Get Payment Data Error Response

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

Update Payment Info 7

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.

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.
3. To avoid double counting the types of order events, specify subsequent order events of the same type as `Other`. For example, if you request Update Payment Info for a `Confirm` order event and subsequently request an Update Payment Info for a payment event of `Authorize`, specify `Other` in the `eventType` of the `OrderInfo` structure and `Authorize` in the `eventType` of the `PayInfo` structure to avoid double counting `Confirm` events. Likewise, when using a `Confirm_COF` order `event type`, in subsequent requests to Update Payment Info for a payment event, specify `Other` in the `eventType` of the `OrderInfo` structure to avoid double counting `Confirm_COF` events.

Related Content

Event Types

Card on File Events

Promotions

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 one or more payment events, you must specify at least one order event, which associates the payment event with an order ID.

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 events or payment event subsequent to a `Confirm` or `Confirm_COF` order event.	`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`

Related Content

Update Payment Info Summary (Parent Topic)

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`

Related Content

Update Payment Info Summary (Parent Topic)

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

Related Content

Update Payment Info Summary (Parent Topic)

Update Payment Info Request

Path and Endpoints

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

Complete endpoint:

Sandbox:

`https://sandbox.api.visa.com/wallet-services-web/payment/data/{callId}?apikey=key`

Live:

`https://api.visa.com/wallet-services-web/payment/info/{callId}?apikey=key`

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

Method

`PUT`

Required Headers

Header	Description
`x-pay-token`	A token identifying the transaction and its contents. **Format:** Alphanumeric; maximum 100 characters in the form of `xv2:`*`UTC_Timestamp`*`:`*`HMAC-SHA256_hash`*, where • *UTC_Timestamp* is a UNIX Epoch timestamp • *HMAC-SHA256_hash* is an HMAC-SHA256 hash using the shared secret associated with your API key and the following unseparated items: 1. Timestamp from the transaction; exactly the same as *UNIX_UTC_Timestamp* 2. Resource path (API name) 3. This HTTPS request's query string, if it exists **Note** To create the query string, concatenate all query string components (names and values) as UTF-8 characters, which are URL-encoded per RFC 3986. Hex characters must be uppercase. Multiple parameters must be sorted using lexicographic byte ordering and separated from each other by an ampersand (`&`) character (ASCII code 38). Parameter names are separated from their values by the = character (ASCII character 61), which must be present even if the value is empty. "Unreserved" characters specified in Section 2.3 of *RFC 3986* , including dash -, dot `.`, underscore _, and tilde ~ should not be URL-encoded. 4. Complete request body, when a request body exists **Example:** `x-pay-token: xv2:1440199445:`*`HMAC-SHA256_hash result`* Since 6.3

`Accept`	Acceptable response format.
	
	Format: Must include the following value:
	
	`application/json`
	
	`application/xml`
	
	Example: `Accept: application/json`
	
	Example: `Accept: application/xml`
	
	Since 2.0

Header	Description

`Content-Type`	Format of the content.
	Format: Must include the following value:
	`application/json`
	`application/xml`
	Example: `Content-Type: application/json`
	Example: `Content-Type: application/xml`
	Since 2.0

Query Parameters

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

Update Payment Info Request Parameters

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`.

Property	Description
`orderInfo`	*(Required)* Order info properties, one per `orderInfo` structure. **Format:** One or more `orderInfo` structures. Since 3.2
`payInfo`	*(Optional)* Pay info properties structure. **Format:** A `payInfo` structure. Since 3.2

Related Content

Order Info Properties

Pay Info Properties

Order Info Properties

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

Property	Description
`total`	*(Required)* 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
`currencyCode`	*(Required)* The currency with which to process the transaction. Required because `total` must be provided. **Format:** It is one of the following ISO 4217 standard alpha-3 code values: • `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) • `INR` - Indian rupee (Since 4.6) • `KWD` - Kuwaitii Dinar (Since 5.1) • `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) • `QAR` - Qatari Riyal (Since 5.1) • `SAR` - Saudi Riyal (Since 5.1) • `SGD` - Singapore Dollar (Since 2.7) • `ZAR` - Rand (Since 2.7) • `UAH` - Ukranian Hryvnia (Since 5.1)

Property	Description
	• `AED` - UAE Dirham (Since 2.7) • `GBP` - UK Pound Sterling (Since 4.3) • `USD` - US Dollar Currency codes must be uppercase. **Example:** `"currencyCode":"USD"` Since 2.0
`subtotal`	*(Required)* 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
`shippingHandling`	*(Optional)* 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
`tax`	*(Optional)* 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
`discount`	*(Optional)* Total of discounts related to the payment. If provided, it is a positive value representing the amount to be deducted from the total. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"discount" : "2.50"` Since 2.0
`giftWrap`	*(Optional)* Total gift-wrapping charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"giftWrap" : "1.99"` Since 2.0

Property	Description
`misc`	*(Optional)* Total uncategorized charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"misc":"1.00"` Since 2.0
`eventType`	*(Required)* Kind of event associated with the update. **Format:** It is one of the following values: • `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 events or payment event subsequent to a `Confirm` or `Confirm_COF` order event. **Example:** `"eventType":"Confirm"` Since 2.0
`orderId`	*(Required)* Merchant's order ID associated with the payment. **Format:** Alphanumeric; maximum 100 characters Since 2.0

Property	Description
promoCode	*(Optional)* Promotion codes associated with the payment. **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 **Example:** `"promoCode": "17.15"` Since 2.0
reason	*(Optional)* Reason for the update. **Format:** Alphanbetic, numeric, spaces, and the following characters: `~ ` ! @ # $ % ^ * ( ) - _ + = [{] } \ | : / ? . , Àà ÂâÄä Çç Èè Éé Êê Ëë Îî Ïï ÑñÔô Ùù Ûû Üü Ÿÿ Æ æ OE oe ° º ª` Maximum 255 characters Since 2.0

Related Content

Multiple Info Properties (Parent Topic)

Pay Info Properties

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

Property	Description
payTransId	*(Optional)* 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. **Format:** Alphanumeric; maximum 100 characters Since 2.0
eventType	*(Required)* Kind of event associated with the update. **Format:** It is one of the following values:

Property	Description
	• `Authorize` • `Capture` • `Refund` • `Cancel` • `Fraud` • `Chargeback` • `Other` **Example:** `"eventType":"Authorize"` Since 2.0
`eventStatus`	*(Required)* Status of the event. **Format:** It is one of the following values: • `Success` • `Failure` • `Fraud` • `Chargeback` • `Other` 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`. **Example:** `"eventStatus":"Success"` Since 2.0
`currencyCode`	*(Required)* The currency with which to process the transaction. Required because `total` must be provided. **Format:** It is one of the following ISO 4217 standard alpha-3 code values: • `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)

Property	Description
	• `INR` - Indian Rupee (Since 4.6) • `KWD` - Kuwaiti Dinar (Since 5.1) • `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) • `QAR` - Qatari Riyal (Since 5.1) • `SAR` - Saudi Riyal (Since 5.1) • `SGD` - Singapore Dollar (Since 2.7) • `ZAR` - Rand (Since 2.7) • `UAH` - Ukranian Hryvnia (Since 5.1) • `AED` - UAE Dirham (Since 2.7) • `GBP` - UK Pound Sterling (Since 4.3) • `USD` - US Dollar Currency codes must be uppercase. **Example:** `"currencyCode":"USD"` Since 2.0
`total`	*(Required)* 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`	*(Optional)* Authorization code associated with the transaction. **Format:** Numeric; maximum 6 digits **Example:** `"authCode":"123456"` Since 2.0

Property	Description
`avsResponseCode`	*(Optional)* Address verification system response code. **Format:**Alphanumeric **Example:** `"avsResponseCode" : "V"` Since 2.0
`reason`	*(Optional)* Reason for the update. **Format:** Alphanumeric; maximum 255 characters Since 2.0

Related Content

Multiple Info Properties (Parent Topic)

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

Update Multiple Info Structure Examples

JSON Request Body

The `orderId` in the `orderInfo`, as described in this example, should match the `orderId` that was passed in the `orderInfo` of the initial `UpdatePayment` call.

```
{
    "updateInfo": [
        {
            "payInfo": {
                "reason": "Just a test",
                "avsResponseCode": "Y",
                "total": "91.00",
                "currencyCode": "USD",
                "eventStatus": "Success",
                "eventType": "Authorize",
                "payTransId": "123456789"
            }
        },
        {
            "orderInfo": {
                "currencyCode": "USD",
                "discount": "25.00",
                "eventType": "Confirm",
                "giftWrap": "0",
                "misc": "0",
                "orderId": "123456",
                "promoCode": "VISACHECKOUT25",
                "reason": "Order placed",
                "shippingHandling": "10.00",
                "subtotal": "100.00",
                "tax": "6.00",
                "total": "91.00"
            }
        }
    ]
}
```

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: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: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
...
/>
```

Order Update Success Example

JSON Request Including Headers

```
PUT
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/info/03e...
?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/03e...
?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 Success Example

JSON Request Including Headers

```
PUT
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/info/03e...
?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/03e...
?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: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: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

JSON Update Payment Info Error Example

```
{ "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. "
}
}
```

XML Update Payment Info Error Example

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

Update Payment Info Pixel Image 8

Update Payment Info Pixel Image Summary

Note

You should use the Update Payment Info API. The Update Payment Info Pixel Image is provided only for legacy integrations.

You can confirm a purchase by including `payment/updatepaymentinfo.gif` on a page that appears after the customer reviews and approves the order. Specifically, you associate parameters that convey the purchase information with the image before the image is loaded on the page, allowing the information to be transmitted to Visa Checkout when the image is loaded.

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`

Related Content

Card on File Events

Promotions

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`

Related Content

Update Payment Info Pixel Image Summary (Parent Topic)

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

Related Content

Update Payment Info Pixel Image Summary (Parent Topic)

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`	*(Required)* Public API key, which is different than the shared secret. **Format:** Alphanumeric; maximum 100 characters **Example:** `apikey=xxxxxxxxxxxxxxxxxxxx` Since 2.0
`callId`	*(Required)* Visa Checkout transaction ID returned by the Visa Checkout `payment.success` event. **Format:** Alphanumeric; maximum 48 characters **Example:** `"callId":"..."` Since 2.0
`currencyCode`	*(Required)* The currency with which to process the transaction. Required because `total` must be provided. **Format:** It is one of the following ISO 4217 standard alpha-3 code values: • `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) • `INR` - Indian rupee (Since 4.6) • `KWD` - Kuwaiti Dinar (Since 5.1) • `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) • `QAR` - Qatari Riyal (Since 5.1) • `SAR` - Saudi Riyal (Since 5.1)

Property	Description
	• `SGD` - Singapore Dollar (Since 2.7) • `ZAR` - Rand (Since 2.7) • `UAH` - Ukranian Hryvnia (Since 5.1) • `AED` - UAE Dirham (Since 2.7) • `GBP` - UK Pound Sterling (Since 4.3) • `USD` - US Dollar Currency codes must be uppercase. **Example:** `"currencyCode":"USD"` Since 2.0
`discount`	*(Optional)* Total of discounts related to the payment. If provided, it is a positive value representing the amount to be deducted from the total. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"discount" : "2.50"` Since 2.0
`eventType`	*(Required)* Kind of event associated with the update. **Format:** It is one of the following values: • `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 **Example:** `"eventType":"Confirm"` Since 2.0
`giftWrap`	*(Optional)* Total gift-wrapping charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"giftWrap" : "1.99"` Since 2.0

Property	Description
misc	*(Optional)* Total uncategorized charges in the payment. **Format:** Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits **Example:** `"misc":"1.00"` Since 2.0
orderId	*(Required)* Merchant's order ID associated with the payment. **Format:** Alphanumeric; maximum 100 characters Since 2.0
promoCode	*(Optional)* Promotion codes associated with the payment. **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 **Example:** `"promoCode": "17.15"` Since 2.0
reason	*(Optional)* Reason for the update. **Format:** Alphanumeric; maximum 255 characters Since 2.0
shippingHandling	*(Optional)* 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	*(Required)* 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

Property	Description
tax	*(Optional)* 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	*(Required)* 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	Visa Checkout error message. **Format:** Alphanumeric **Example:** `"v-message":"callid ff6e689c-170c-4f18-a1ba-da5047a35f152 was not found"` Since 2.0

Update Payment Info Request Inside an Image Tag

```
<img src="https://sandbox.secure.checkout.visa.com/wallet-services-web
/payment/updatepaymentinfo.gif?apikey=...&callId=...
&currencyCode=USD&eventType=Confirm&total=11.00&subtotal=11.00" />
```

Update Payment Info Request

```
GET https://sandbox.secure.checkout.visa.com
```

```
/wallet-services-web/payment/updatepaymentinfo.gif
?apikey=...
&callId=...
&currencyCode=USD
&eventType=...
&discount=...
&giftWrap=...
&misc=...
&orderId=...
&promoCode=...
&reason=...
&subtotal=...
&shippingHandling=...
&tax=...
&total=...
```

Update Payment Info Error Response

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

Decrypting Consumer Information A

Decrypting Consumer Information Introduction

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

Visa Checkout supports the use of xv2 API keys for `apikey` in JavaScript (`V.init`) and in API requests, which require an `x-pay-token` header. Creation of the xv2 token requires a different algorithm; however, there is no change to API key usage for making requests.

In addition to the change of API keys, you need to provide an \encryption key, `encryptionKey`, which Visa Checkout uses to encrypt data in the consumer information payload using the shared secret associated with this encryption key.

Contact your Visa Checkout representative to get set up for using these keys in your Visa Checkout projects

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 (different than the encryption 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 (different than the encryption key).

1. Decrypt the dynamic key, which is different than the encryption 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 the shared secret associated with your encryption 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 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.

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", "..."));
}
}
```

Node.js Decryption Example

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

```
const crypto = require('crypto');

module.exports = {
    decryptPayload: function(key,wrappedKey,payload){
        let decryptedKey = decrypt(wrappedKey,key);
        let decryptedMsg = decrypt(payload,decryptedKey);
        return decryptedMsg.toString('utf8');
    }
}

function decrypt(encrypted,key){
    let encryptedBuffer = new Buffer(encrypted,'base64');
    // TODO: Check that data(encryptedBuffer) is at least bigger
    // than HMAC + IV length , i.e. 48 bytes
    let hmac = new Buffer(32);
    let iv = new Buffer(16);
    encryptedBuffer.copy(hmac,0,0,32);
    encryptedBuffer.copy(iv,0,32,48);
    let data = Buffer.from(encryptedBuffer).slice(48);

    var hash = crypto.createHmac('SHA256', key).update
    (Buffer.concat([iv,data])).digest();
    if(!hmac.equals(hash)){
        // TODO: Handle HMAC validation failure
        return '';
    }

    let decipher = crypto.createDecipheriv('aes-256-cbc',
    crypto.createHash('sha256').update(key).digest(), iv);
    let decryptedData = Buffer.concat([decipher.update(data), decipher.final()]);
    return decryptedData;
}
```

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
```

HMAC-SHA256–Bit Hashing B

About the HMAC-SHA256–Bit Hashing Algorithm

HMAC-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, HMAC-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 HMAC-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 HMAC-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.

Important

Because the examples use the shared secret, do not run it from a web page. Only execute the sample from a protected server.

HMAC-SHA256 Hash Algorithm in PHP Example

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

```
<?php

class XPayTokenGenerator {
    /**
     * X-PAY-TOKEN generator for version 2 (xv2:)
     *
     * @param sharedSecret
     * @param resourcePath
     *              Example: payment/info/{callId}
     * @param queryString
     *              Example: apiKey=ABCD....XYZ
     * @param requestBody
     *              Example:
{"orderInfo":{"currencyCode":"USD","eventType":"Confirm","orderId"
   :"testorderID","subtotal":"80.1","total":"101"}}
     * @return
     */
    public function generateXPayTokenV2($sharedSecret, $resourcePath
      , $queryString, $requestBody) {
        $timestamp = time();
       $beforeHash =
$timestamp.$resourcePath.$queryString.$requestBody;
```

```
        $xPayToken = "xv2:".$timestamp.":".hash_hmac('sha256', $beforeHash
          , $sharedSecret);
        return $xPayToken;
      }
  }
?>
```

HMAC-SHA256 Hash Algorithm in Python Example

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

```
from calendar import timegm
from datetime import datetime
from hashlib import sha256
import hmac

def _get_x_pay_token(shared_secret, resource_path, query_string, body):
    timestamp = str(timegm(datetime.utcnow().timetuple()))
    pre_hash_string = timestamp + resource_path + query_string + body
    hash_string = hmac.new(shared_secret,
                           msg=pre_hash_string,
                           digestmod=sha256).hexdigest()
    return 'xv2:' + timestamp + ':' + hash_string
```

HMAC-SHA256 Hash Algorithm in Java Example

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

```
import java.math.BigInteger;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.SignatureException;


public static String generateXpaytoken(String resourcePath, String queryString
   , String requestBody, String sharedSecret) throws SignatureException {
    String timestamp = timeStamp();
    String beforeHash = timestamp + resourcePath + queryString + requestBody;
    String hash = hmacSha256Digest(beforeHash, sharedSecret);
    String token = "xv2:" + timestamp + ":" + hash;
    return token;
}

private static String timeStamp() {
        return String.valueOf(System.currentTimeMillis()/ 1000L);
    }
private static String hmacSha256Digest(String data, String sharedSecret)
        throws SignatureException {
    return getDigest("HmacSHA256", sharedSecret, data, true);
}
private static String getDigest(String algorithm, String sharedSecret, String data
   , boolean toLower) throws SignatureException {
    try {
        Mac sha256HMAC = Mac.getInstance(algorithm);
```

```
        SecretKeySpec secretKey = new SecretKeySpec(sharedSecret.getBytes
           (StandardCharsets.UTF_8), algorithm);
        sha256HMAC.init(secretKey);

        byte[] hashByte = sha256HMAC.doFinal(data.getBytes(StandardCharsets.UTF_8));
        String hashString = toHex(hashByte);

        return toLower ? hashString.toLowerCase() : hashString;
    } catch (Exception e) {
        throw new SignatureException(e);
    }
}

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

HMAC-SHA256 Hash Algorithm in Ruby Example

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

```
def get_xpay_token(shared_secret, resource_path, query_string, request_body)
require 'digest'
timestamp = Time.now.getutc.to_i.to_s
hash_input = timestamp + resource_path + query_string + request_body
hash_output = OpenSSL::HMAC.hexdigest(OpenSSL::Digest.new('sha256')
   , shared_secret, hash_input)
return "xv2:" + timestamp + ":" + hash_output
end
```

HMAC-SHA256 Hash Algorithm in C# Example

The following C# example shows how to create the `x-pay-token` header:

```
private static string getTimestamp() {
    long timeStamp = ((long) DateTime.UtcNow.Subtract(new DateTime(1970
       , 1, 1, 0, 0, 0, DateTimeKind.Utc)).TotalMilliseconds) / 1000;
    return timeStamp.ToString();
}

private static string getHash(string data) {
    var hashString = new HMACSHA256(Encoding.ASCII.GetBytes(SHARED_SECRET));
    var hashbytes = hashString.ComputeHash(Encoding.ASCII.GetBytes(data));
    string digest = String.Empty;

    foreach (byte b in hashbytes) {
        digest += b.ToString("x2");
    }

    return digest;
}

private static string getXPayToken(string resourcePath, string queryString
   , string requestBody) {
```

```
    string timestamp = getTimestamp();
    string sourceString = timestamp + resourcePath + queryString  + requestBody;
    string hash = getHash(sourceString);
    string token = "xv2:" + timestamp + ":" + hash;
    return token;
}
```

Clickjacking Prevention C

Clickjacking Prevention Steps

To prevent clickjacking of your pages, each page must contain JavaScript to verify that there are no transparent layers, such as might be the case if your page was loaded as an iFrame of a page containing malicious code, and that only your site can load your pages.

Related Content

Checking for Hidden Layers

Using the X-Options Header

Testing Your Clickjacking Prevention Implementation

Checking for Hidden Layers

Pages that prevent clickjacking contain JavaScript, such as the following, to verify that there are no transparent layers in which malicious code could reside:

```
<head>
...
<style id="antiClickjack">body{display:none;}</style>
<script type="text/javascript">
if (self === top) {
var antiClickjack = document.getElementById("antiClickjack");
antiClickjack.parentNode.removeChild(antiClickjack);
} else {
top.location = self.location;
}
</script>
...
</head>
```

Related Content

Clickjacking Prevention Steps (Parent Topic)

Using the X-Options Header

Messages directed at your pages must include an `X-FRAME-OPTIONS` header to verify that the response is known to be from your web application:

- `X-FRAME-OPTIONS DENY` prevents anything from framing your page.
- `X-FRAME-OPTIONS SAMEORIGIN` prevents anything except your application from framing your page.

In addition, you should use the frame-ancestors directive in a Content-Security-Policy HTTP response header to indicate whether or not a browser should be allowed to render a page in a `frame` or `iframe`. If supported by the consumer's browser, sites can also use this directive to avoid clickjacking attacks by ensuring that their content is not embedded into other sites.

Related Content

Clickjacking Prevention Steps (Parent Topic)

Testing Your Clickjacking Prevention Implementation

To test your implementation of anti-clickjacking measures:

> **Note**
>
> These steps assume your site is not already in an iFrame.

1. Install or use a test server that is not being used for your production or sandbox site and does not contain the pages that you want to test. For example, you can test using Tomcat on `localhost:8080`.
2. Create a page on your test server that loads the page containing the Visa Checkout button in an iFrame.

```
<html>
<body>
<iframe src="https://www.yoursite.com/..." width=100% height=100%>
  <p>Your browser does not support iframes.</p>
</iframe>
</body>
</html>
```

3. Test the page you created to load your actual page in an iFrame.

As a best practice, you should automate these steps so that you automatically run a script to test your clickjacking prevention measures whenever you change or add a page to your site.

Related Content

Clickjacking Prevention Steps (Parent Topic)

Example Server-Side Clickjacking Prevention Implementation

The following example shows how to implement `X-FRAME-OPTIONS DENY` or `X-FRAME-OPTIONS SAMEORIGIN` headers in a Java servlet for pages served by Tomcat:

Related Content

Java Servlet

Tomcat Configuration

Java Servlet

The following sample implements a servlet to provide an X-Frame-Options and Content-Security-Policy frame-ancestors header as a filter:

```
package com.testvco.filter;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletResponse;
```

```
public class ClickjackFilter implements Filter{
 private String xFrameOptions = "DENY";
 private String frameAncestors = "frame-ancestors '_VALUE_'";

 @Override
 public void destroy() {
 }

 @Override
 public void doFilter(ServletRequest request, ServletResponse response,
   FilterChain chain) throws IOException, ServletException {
  HttpServletResponse res = (HttpServletResponse)response;
  res.addHeader("X-Frame-Options", xFrameOptions);
  res.addHeader("Content-Security-Policy", frameAncestors);
  chain.doFilter(request, response);
 }

 @Override
public void init(FilterConfig filterConfig) throws ServletException {
String xFrameOptions = filterConfig.getInitParameter("xFrameOptions");
String frameAncestors=filterConfig.getInitParameter("frameAncestors");
  if ( xFrameOptions != null ) {
   this.xFrameOptions = xFrameOptions;
  }
  if ( frameAncestors != null ) {
   this.frameAncestors
    = this.frameAncestors.replace("_VALUE_", frameAncestors);
  }else{
   this.frameAncestors
    = this.frameAncestors.replace("_VALUE_", "none");
  }
 }
}
```

Related Content

Example Server-Side Clickjacking Prevention Implementation (Parent Topic)

Tomcat Configuration

Add the filter definition and mapping to your web application's `web.xml` file. Set up the mapping so that it applies to any page that hosts the Visa Checkout button:

```
 <?xml version=”1.0” encoding=”UTF-8”?>
 <web-app ...>
 ...
 <!-- Clickjack prevention -->
  <!-- Use 'DENY' or 'SAMEORIGIN' or 'ALLOW-FROM http....' -->
  <!-- for xFrameOptions param -->
  <!-- Use 'none' or 'self' or 'http...' for frameAncestors param -->
  <!-- for frameAncestors param -->
  <filter>
   <filter-name>ClickjackFilter </filter-name>
   <filter-class>com.testvco.filter.ClickjackFilter </filter-class>
   <init-param>
    <param-name>xFrameOptions </param-name>
    <param-value>DENY </param-value> <!-- Change as needed -->
   </init-param>
```

```
   <init-param>
    <param-name>frameAncestors </param-name>
    <param-value>none </param-value> <!-- Change as needed -->
   </init-param>
  </filter>
  <filter-mapping>
   <filter-name>ClickjackFilter </filter-name>
   <url-pattern>/* </url-pattern>
  </filter-mapping>
 ...
 </web-app>
```

Related Content

Example Server-Side Clickjacking Prevention Implementation (Parent Topic)

AVS and CVV Responses D

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** 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.
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.

AVS Code	Description
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.
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.

AVS Code	Description
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.
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.

CVV Code	Description
U	Card verification is not supported by the issuing bank.
X	Card verification is not supported by the card association.

Branding Requirements E

Visa Checkout Buttons

You can place the following Visa Checkout buttons on your web page.

Assets: Visa Checkout Buttons

Button	Description
	Standard button without card art
	Neutral button without card art
	Button with card art

Visa Checkout Dynamic Acceptance Marks

You can use the Visa Checkout dynamic acceptance marks to let consumers know that you accept Visa Checkout on your pages. However, not all sizes and colors of acceptance marks can be dynamic. The dynamic acceptance marks include versions blue 01, blue 02, white 01, and white 02 in sizes 99x34, 49x31, and 40x30. Size 28x21 acceptance marks and disabled dynamic marks cannot be dynamic. You can choose the style, color, and size based on the needs of your site. However, you must link to the dynamic acceptance mark.

Note

Do not create your own dynamic acceptance marks. You can only use them if you link to the associated URL, which Visa Checkout provides.

The base URL is https://assets.secure.checkout.visa.com/VCO/images/. It is followed by the .png graphic; for example, acc_99x34_wht01.png. The complete URL is https://assets.secure.checkout.visa.com/VCO/images/.png.

The following tables list all available Visa Checkout acceptance marks when enabled and disabled.

Assets: Visa Checkout Dynamic Acceptance Marks When Enabled

The following table lists all available dynamic acceptance marks.

White on Blue	Blue	Blue on Neutral	White
			
			

Assets: Visa Checkout Dynamic Acceptance Marks When Disabled

The following table lists available disabled acceptance marks.

Disabled with background	**Disabled** without background
	
	
	

Revision History

- Version 2.0, April 29, 2014
- Version 2.1, June 10, 2014
- Version 2.2, July 8, 2014
- Version 2.3, August 5, 2014
- Version 2.4, September 2, 2014
- Version 2.5, October 7, 2014
- Version 2.6, November 11, 2014
- Version 2.7, January 27, 2015
- Version 2.8, March 31, 2015
- Version 2.9, April 28, 2015
- Version 3.0, May 29, 2015
- Version 3.1, June 30, 2015
- Version 3.2, July 28, 2015
- Version 3.3, August 25, 2015
- Version 3.4, September 29, 2015
- Version 3.5, October 27, 2015
- Version 3.6, January 26, 2016
- Version 3.8, March 22, 2016
- Version 3.9, April 19, 2016
- Version 4.1, June 21, 2016
- Version 4.2, July 19, 2016
- Version 4.3, August 30, 2016
- Version 4.4, September 20, 2016
- Version 4.5, October 25, 2016
- Version 4.6, January 24, 2017
- Version 4.7, February 28, 2017
- Version 5.0, May 2, 2017
- Version 5.1, May 24, 2017
- Version 5.2, June 28, 2017
- Version 5.3, July 26, 2017
- Version 5.4, August 23, 2017
- Version 5.5, September 27, 2017
- Version 5.6, November 1, 2017
- Version 5.7, December 7, 2017
- Version 5.8, January 24, 2018

- Version 6.0, March 28, 2018
- Version 6.4, July 25, 2018
- Version 6.5, August 22, 2018
- Version 6.6, September 26, 2018
- Version 6.7, October 31, 2018