Visa Checkout

JavaScript Integration Guide

Effective: October 7, 2014

Important Note on Intellectual Property

Contents

THIS PAGE INTENTIONALLY LEFT BLANK.

Integration Overview 1

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 existing 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 only a few requirements for its use.

About the Visa Checkout Button and Lightbox

Checking out and paying with Visa Checkout begins with a button click on the **Visa Checkout** button, which could be a generic version or one that shows the image of the credit card brand being proposed to the consumer:

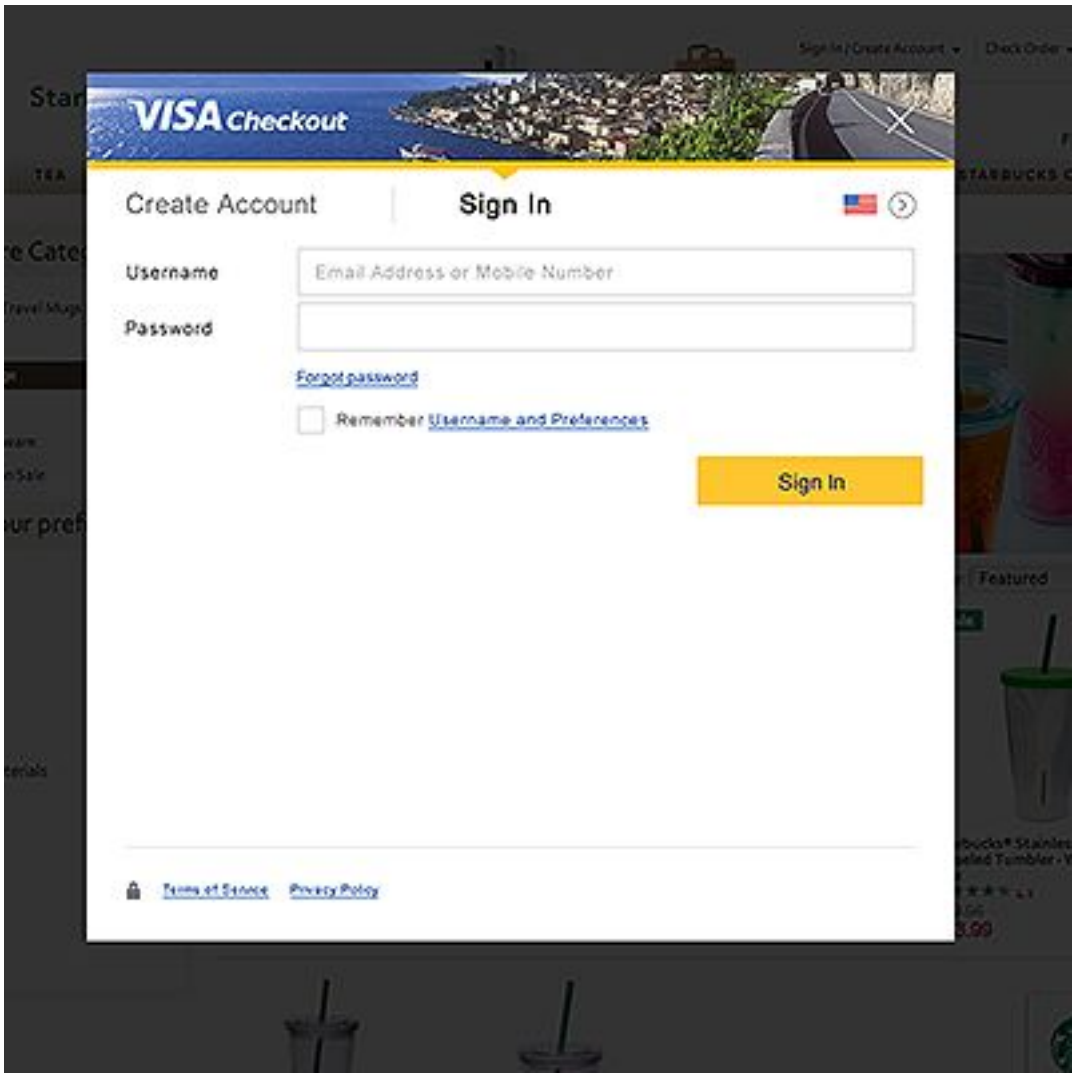
- 

- 

Important:

You must follow the Visa Checkout user interface guidelines, which are described in *Getting Started With Visa Checkout*.

Regardless of how the consumer arrives at a page with a Visa Checkout button, when a consumer clicks a **Visa Checkout** button, the Visa Checkout lightbox appears from which the consumer can either sign up to create an account, sign in, and make a payment:



Recommended Browser Versions

The following browsers are recommended for use by Visa Checkout:

- Internet Explorer, version 8 or later

  Note: *Do not use the compatibility setting; specifically, do not use it to specify a version less than IE8; for example, do not specify* `x-ua-compatible=IE7` *in your pages.*

- Firefox, version 25 or later,
- Chrome, version 31 or later, excepting beta versions
- Safari, version S6 or later
- iOS, version 6 or later
- Android, version 4 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.

Checkout Flow

The following diagram, explained below, shows a typical frontend checkout flow, which identifies the points of interaction between your site and Visa Checkout:

Typically, a consumer decides to click **Visa Checkout** to check out from either your shopping cart page or on your payment page. Checkout or payment from other pages is possible; in all cases, you must place a **Visa Checkout** button on any page from which the consumer can check out or pay. Each of the pages on which you place a **Visa Checkout** button must load the Visa Checkout library, which causes Visa Checkout to send an initialization event when the page is loaded. You respond to the event by setting Visa Checkout lightbox characteristics related to its operation and appearance and by setting characteristics about the payment request itself.

Clicking a **Visa Checkout** button invokes the Visa Checkout lightbox, enabling a consumer to sign into Visa Checkout and make a payment request. After signing in to Visa Checkout, the consumer can change the payment method and, if enabled to do so, change the shipping address. When the consumer finishes and the lightbox closes, or an error occurs, Visa Checkout sends you an event that includes status information and a *call ID* that identifies the payment request.

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. See Complete Web Page HTML Example for an example web page that contains the JavaScript function.

In Step 2, a payment success event returns encrypted consumer payment information, which includes card verification, authentication, and risk information. The personal 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; see Integration Options for more information.

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 eCommerce 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.

Integration Options

Visa Checkout provides you with several options to handle consumer payment information returned by a payment event and update Visa Checkout with the result of processing a payment. You can handle 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; see Visa Checkout API Summary for more information
- Passing the ID associated with the Visa Checkout payment request, which is a *call ID* in Visa Checkout terminology, to your server or to a processor or an eCommerce partner for payment processing

  Note: *You can pass the call ID to your 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.

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.

Important:

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. For more information, see *Visa Checkout Client API Reference*.

Note: *See* Visa Checkout Client API Reference *for consumer information field contents.*

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 eCommerce partner. The following integration options are available, depending on your configuration:

- Update Visa Checkout payment information from your web page directly, by passing parameters when a 1-pixel image provided by Visa Checkout is loaded; see Update Payment Info Pixel Image
- Pass the call ID to your server or to the entity that will take action to on your behalf; in which case, the call ID can be used with the Update Payment Info API to update Visa Checkout payment information
- Call a Visa Checkout API, Update Payment Info, to update payment information, which is also what an entity that takes this action for you must do; for more information, see *Visa Checkout Client API Reference*

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 a particular programming language to use its API.

Important:

Visa Checkout does not require you to call Visa Checkout APIs because all interactions with Visa servers can also be handled from your pages using only the Visa Checkout JavaScript library. Specifically, you do not need to support domain based filtering nor do you need access to specific Visa IP addresses.

The following APIs are available to all merchants:

Resource	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/info/{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.

Note: *Additional APIs are available for onboarding merchants for Visa Checkout and managing their relationship with those merchants. See the* Visa Checkout Client API Reference *for all API information, including consumer payment information returned by the Visa Checkout API or payment success event.*

Clickjacking Prevention Requirement

You must provide code on each page that hosts a Visa Checkout button and headers on your server to prevent clickjacking, which could occur if malicious code is hidden beneath legitimate buttons or other clickable content on your web page. For example, malicious code might monitor keystrokes and steal confidential information. Customers could be "clickjacked" when clicking a legitimate link on an infected page in which there are actually buttons on a transparent layer they cannot see.

Visa Checkout periodically reviews each page from which a Visa Checkout button is clicked to determine if adequate anticlickjacking preventions have been taken. To prevent clickjacking, you must ensure that pages cannot be loaded as an iFrame of some other page. Specifically, you must

- Ensure that the associated DOM document for the page has no child pages in which malicious code could reside.
- Implement X-FRAME-OPTIONS DENY or X-FRAME-OPTIONS SAMEORIGIN filtering for headers on your server to prevent your page from being loaded from another domain.

For more information about clickjacking prevention, see Clickjacking Prevention.

Fraud and Risk

Visa Checkout uses a combination of proprietary and third-party technologies to implement fraud checks while processing transactions on your behalf. These checks provide account validations on all Visa Checkout accounts when :

- the account is created or accessed
- a customer logs in to Visa Checkout
- a card is associated with the account, updated, or used in a transaction

Visa Checkout Fraud Checks

Examples of fraud checks include device and IP data checks, velocity, address verification (AVS), and card number verification (CVN or CVV) results from the card issuer, enrollment attributes, Visa Checkout transaction history, and Visa internal fraud checks. In addition, Visa Checkout provides risk advice, which categorizes the transaction based on anticipated risk. Specifically, for every card added to a Visa Checkout account, regardless of card brand, Visa Checkout performs a validation procedure prior to passing

the card information to a merchant. This validation procedure includes an AVS check (Address Verification) and a verification of the CVN or CVV number. A full or partial match is required for Address Verification and a match or unsupported response is required for CVN.

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 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.**

Visa Checkout Risk Advice

When assessing a transaction for anticipated risk, Visa Checkout categorizes a transaction as:

- Low anticipated risk
- Medium anticipated risk
- High anticipated risk

Because fraud experience and tolerance for risk vary from merchant to merchant, you should calibrate Visa Checkout risk advice with your own experience and with your customers' characteristics and behavior to use it effectively in your own fraud management policies. For example, if you sell low-price recurring digital content, you may be more willing to accept risk because the financial exposure is much less, and your customers leave you few alternative mitigation strategies, than if you sell high priced, low margin goods.

To use Visa Checkout risk advice with your existing system, consider performing the following analysis:

- Incorporate the Visa Checkout risk advice into your fraud management policies to make decisions, paying particular attention to transactions with a high risk indicator
- Monitor your transactions for at least a month, collecting both your existing system's risk information and Visa Checkout risk advice.
- Compare the results, by type of payment instrument, against actual fraud losses.
- Make appropriate rules changes within your own system to weigh the Visa Checkout risk advice according to the results and your tolerance for risk, lost sales, and manual review costs.

Your analysis may verify that Visa Checkout risk advice confirms your own experience with risk, or you may find that the Visa Checkout categories indicate a different amount of risk for you than the labels imply. Regardless, after calibration with your data, Visa Checkout risk advice can provide another data point for identifying risk, which could be used in a model that triggers an investigative process or procedure.

Risk Declines

Risk 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.

Card Security Code Usage

Visa Checkout performs a verification of the Card Security Code for each card used for a Visa Checkout transaction or passed to a merchant for processing. Similar to a "card on file" scenario, the validation can be 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 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 regards to risk management for Visa Checkout transactions as you would for any other e-Commerce transaction. Because the 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 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 chargeback may depend on the card brand. Merchants are encouraged to speak directly with their acquirer to understand the chargeback re-presentment rules and reversal criteria for a specific card brand.

Visa Checkout JavaScript and Button Reference 2

sdk.js JavaScript Library

Use the `sdk.js` JavaScript library to control the operation of Visa Checkout on your site. There is one version for sandbox testing and one for live:

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

Example

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

v-button Image Class

Use the `v-button img` class to render a Visa Checkout button that a consumer clicks to initiate a payment. The rendered buttons must follow the Visa Checkout user interface guidelines, which are described in *Getting Started With Visa Checkout*.

Platform	Source URL
Sandbox	`https://sandbox.secure.checkout.visa.com/wallet-services-web/xo/button.png`
Live	`https://secure.checkout.visa.com/wallet-services-web/xo/button.png`

Query Parameters

Parameter	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 `476` and less than `659` if `height` is `47`; default value is `213` • greater than `658` 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

Parameter	Description
`locale`	*(Optional)* The locale, which controls how text displays in a Visa Checkout button and the Visa Checkout lightbox. If not specified, the Accepted-Language value in HTTPS header is used, or if not present, `en_US` is used. **Format:** It is one of the following values: • `en_US` - American English • `en_CA` - Canadian English • `fr_CA` - Canadian French • `en_AU` - Australian 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. **Format:** Comma-separated list of one or more of the following brands: • `VISA` • `MASTERCARD` • `AMEX` • `DISCOVER` **Example:** `cardBrands=VISA,AMEX` Since 2.0
`acceptCanadianVisaDebit`	Whether a Canadian merchant accepts Visa Canada debit cards; required for Canadian merchants, otherwise, ignored. **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

Examples

```
<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?..."
/>
```

Tell Me More Link

Use the `v-learn-default` or `v-learn` `<a>` (hyperlink) class to provide a **Tell Me More** link that a consumer clicks to learn more about Visa Checkout. The link is described in *Getting Started With Visa Checkout*. These classes cause a pop up to be displayed in the specified language, which by default is `en_US`;

Attribute	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: • `en_US` - American English (default) • `en_CA` - Canadian English • `fr_CA` - Canadian French • `en_AU` - Australian English Since 2.5

You must provide the link's text, which typically is **Tell Me More**, in the specified locale. The `v-learn-default` class provides default styling, e.g. color, font, and size, and right-aligns the text to the **Visa Checkout** button's container, not to the button itself. Use the `v-learn` class if you need a custom style or position.

Examples

```
<a href="#" class="v-learn-default">Tell Me More</a>
<a href="#" class="v-learn" data-locale=”en_US”>Tell Me More</a>
```

onVisaCheckoutReady Function

You control Visa Checkout button and lightbox operation by defining an `onVisaCheckoutReady` function that includes handlers for initialization and purchase events. The function includes 2 kinds of 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. Specify actions to take on the following Visa Checkout events: • `payment.success` • `payment.cancel` • `payment.error` Since 2.0

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>
```

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. You specify values for these properties:

Property	Description
`apikey`	*(Required)* The API key that Visa Checkout created when you created the Visa Checkout account. You will use both a live key and a sandbox key, which are different from each other. **Format:** Alphanumeric; maximum 100 characters Since 2.0
`externalProfileId`	*(Optional)* Profile ID, and also the profile's name, 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 Since 2.0

Property	Description
`externalClientId`	Unique ID associated with the client, such as a merchant, which could be assigned by you or Visa Checkout. **Format:** Alphanumeric; maximum 100 characters Since 2.0
`sourceId`	*(Optional)* Your merchant reference ID. **Format:** Alphanumeric; maximum 100 characters Since 2.0
`settings`	*(Optional)* One or more name-value pairs, each of which specifies a configuration attribute. **Format:** `settings` Since 2.0
`paymentRequest`	*(Optional)* One or more name-value pairs, each of which specifies a payment request attribute. **Format:** `paymentRequest` Since 2.0

Settings Properties

Property	Description
`locale`	*(Optional)* Override value for the locale, which controls how text displays in the Visa Checkout checkout button and lightbox. By default, Visa Checkout determines the locale from the consumer's browser settings. Do not use the `locale` attribute unless explicit control over the button or lightbox locale is required. **Format:** It is one of the following values: • `en_US` - American English • `en_CA` - Canadian English • `fr_CA` - Canadian French • `en_AU` - Australian 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:** It is one of the following values: • `US` - United States • `CA` - Canada • `AU` - Australia The value of the `country` attribute must be compatible with the value of the `locale` attribute. Since 2.0
`logoUrl`	*(Optional)* Absolute secure (HTTPS) URL path to the logo to display in the Visa Checkout lightbox; otherwise, the default Visa Checkout logo appears. Your image must not exceed 174 pixels in width and should be 34 pixels high; oversize logos will be scaled to fit. **Format:** HTTPS URL; maximum 256 characters 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:** Alphanumeric Since 2.0
`websiteUrl`	*(Optional)* Complete URL to your website. **Format:** Valid URL, beginning with `HTTP` Since 2.0

Property	Description
`customerSupportUrl`	*(Optional)* Your complete customer service or support URL. **Format:** Valid URL, beginning with `HTTP` Since 2.0
`shipping`	*(Optional)* Shipping properties associated with the lightbox; see Shipping Properties. **Format:** `shipping` Since 2.0
`review`	*(Optional)* Review properties associated with the lightbox; see Review Properties. **Format:** `review` Since 2.0
`payment`	*(Optional)* Payment method properties associated with the lightbox; see Payment Properties. **Format:** `payment` Since 2.0
`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. **Format:** It is one of the following values: • `SUMMARY` - Summary information (default) • `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

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: "MegaCorp",
  logoUrl: "www.Some_Image_URL.gif",
  websiteUrl: "www.MegaCorp.com",
  customerSupportUrl: "www.MegaCorp.support.com",
  ...
  dataLevel: "FULL"
 ...
);
```

Shipping Properties

Property	Description
`acceptedRegions`	*(Optional)* Override value for shipping region country codes in the merchant's external profile, which limits selection of eligible addresses in the consumer's account. **Format:** A list of ISO-3166-1 alpha-2 standard codes, such as `US` or `CA`. Since 2.0
`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

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"
  },
 ...
);
```

Review Properties

Property	Description
`message`	*(Optional)* Your message to display on the **Review** page. You are responsible for translating the message. **Format:** Alphanumeric; maximum 120 characters 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 value for `total` must be specified; otherwise **Continue** will be displayed.* Since 2.0

Review Options Configuration Example

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

```
review: {
 message: "Merchant defined message",
 buttonAction: "Continue"
},
```

Payment Properties

Property	Description
`cardBrands`	*(Optional)* Card brands that are accepted. **Format:** Array containing one or more of the following brands: • `VISA` • `MASTERCARD` • `AMEX` • `DISCOVER` Since 2.0
`acceptCanadianVisaDebit`	*(Optional)* Override of whether a Canadian merchant accepts Visa Canada debit cards; ignored for non-Canadian merchants. **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

Payment Options Configuration Example

You can limit the kind of cards you accept:

```
V.init({
 ...
 settings: {
  ...
  payment: {
   cardBrands: [
    "VISA",
    "MASTERCARD"],
   acceptCanadianVisaDebit : "true"
  },
  ...
 },
...
);
```

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: • `USD` - US dollars • `CAD` - Canadian dollars • `AUD` - Australian dollars Currency codes must be uppercase. **Example:** `"currencyCode" : "USD"` Since 2.0
`subtotal`	*(Required)* Subtotal of the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"subtotal" : "9.00"` Since 2.0
`shippingHandling`	*(Optional)* Total of shipping and handling charges in the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"shippingHandling" : "3.00"` Since 2.0

Property	Description
tax	*(Optional)* Total tax-related charges in the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 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 7 digits with optional decimal point and 2 decimal digits **Example:** `"discount" : "2.50"` Since 2.0
giftWrap	*(Optional)* Total gift-wrapping charges in the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"giftWrap" : "1.99"` Since 2.0
misc	*(Optional)* Total uncategorized charges in the payment. **Format:** Numeric; maximum 20 digits, including an optional decimal point. **Example:** `"misc" : "1.00"` Since 2.0
total	*(Optional)* Total of the payment including all amounts. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 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

Property	Description
promoCode	*(Optional)* Promotion codes associated with the payment. Multiple promotion codes are separated by a period (.). **Format:** Alphanumeric; maximum 100 characters **Example:** promoCode: "ABC"."DEF"."XYZ" 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

Note: *The sandbox sets specific risk advice and score values based on the subTotal of the transaction in the request. The risk information is returned in the consumer information payload (encPaymentData) of a successful response.*

Value of the transaction (USD only)	Value of advice in the response	Value of score in the response
0.01 to 0.99 ***Note:*** *Amounts between 0.01 and 0.89 cause the transaction to be declined.*	UNAVAILABLE	0
1.00 to 49.99	LOW	10
50.00 to 149.99	MEDIUM	45
150.00 or more	HIGH	90

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

Response to Payment Success Events

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

Property	Description
`callId`	Visa Checkout transaction ID, which represents the purchase. **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":"..."` Since 2.0

Property	Description
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. **Format:** Alphanumeric; maximum 1024 characters **Example:** "encPaymentData":"..." Since 2.0
externalClientId	Merchant ID of the merchant receiving the payment. **Format:** Alphanumeric; maximum 100 characters Since 2.0
partialShippingAddress	Partial shipping address of the consumer. **Format:** Partial shipping address structure Since 2.0

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

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

Response to Payment Cancelled Events

Property	Description
`callId`	Visa Checkout transaction ID, which represents the cancelled payment. **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

Example

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

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>
```

Complete 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: "...",
     sourceId: "Merchant Defined Source ID",
     settings: {
      locale: "en_US",
      countryCode: "US",
      displayName: "...Corp",
      logoUrl: "www.Some_Image_URL.gif",
      websiteUrl: "www....Corp.com",
      customerSupportUrl: "www....Corp.support.com",
      shipping: {
       acceptedRegions: ["US", "CA"],
```

```
       collectShipping: "true"
      },
      payment: {
       cardBrands: [
        "VISA",
        "MASTERCARD"],
       acceptCanadianVisaDebit: "true"
      },
      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: "Megacorp 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) { ... });
}
 </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>
```

Update Payment Info Pixel Image 3

Update Payment Info Pixel Image Summary

You must 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.

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`)
- A token (`token`) that comprises your shared secret, timestamp, and other information to guarantee the integrity of the parameters being sent to Visa Checkout.
- 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. For the Update Payment Info API, or the Get Payment Data API, see *Visa Checkout Client API Reference*.

Public and Private Key Security

You must follow these rules to implement essential security controls:

- At a high-level, the security of server communications is provided by the use of public-private key pairs. You communicate to Visa Checkout with your public *API key*. You include this key's *shared secret* (private API key), along with data that needs to

be protected from tampering, all of which are encrypted using an SHA256-bit hashing algorithm.

Important:

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

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

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)* Your public API key, which is different than your shared secret. **Format:** Alphanumeric; maximum 100 characters **Example:** `apikey=xxxxxxxxxxxxxxxxxxx` 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
`token`	*(Required)* A token identifying the transaction and its contents. **Format:** Alphanumeric; maximum 100 characters in the form of `token: x:`*`UNIX_UTC_Timestamp`*`:`*`SHA256_hash`*, where • *UNIX_UTC_Timestamp* is a UNIX Epoch timestamp • *SHA256_hash* is an SHA256 hash of the following unseparated items: 1. Your shared secret 2. Timestamp from the transaction; exactly the same as *UNIX_UTC_Timestamp* 3. Resource path (API name) 4. This HTTPS request's query string ***Note:*** *The query string includes one or more parameters in name-value pair format, whose names are separated from values by equal signs (=); an empty value may be omitted but the name and equal sign must be present. The initial question mark (?) is not included.* *All parameters must be present. The parameters must be in lexicographic sort order (UTF-8, uppercase hex characters) with parameters separated from each other by an ampersand (&).* *The query string must be URL encoded (excepting the following characters, per RFC 3986: hyphen, period, underscore. and tilde).* **Example:** `token: x:1358092911:....` Since 2.0

Property	Description
`total`	*(Optional)* Total of the payment including all amounts. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"total":"9.00"` Since 2.0
`currencyCode`	The currency with which to process the transaction. Required if `total` is provided. **Format:** It is one of the following ISO 4217 standard alpha-3 code values: • `USD` - US dollars • `CAD` - Canadian dollars • `AUD` - Australian dollars Currency codes must be uppercase. **Example:** `"currencyCode":"USD"` Since 2.0
`subtotal`	*(Required)* Subtotal of the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"subtotal":"9.00"` Since 2.0
`shippingHandling`	*(Optional)* Total of shipping and handling charges for the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"shippingHandling" : "3.00"` Since 2.0
`tax`	*(Optional)* Total tax-related charges in the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"tax":"1.00"` Since 2.0
`discount`	*(Optional)* Total of discounts related to the payment. If provided, it is subtracted from the `subtotal`. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"discount" : "-2.50"` Since 2.0

Property	Description
giftWrap	*(Optional)* Total gift-wrapping charges in the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 decimal digits **Example:** `"giftWrap" : "1.99"` Since 2.0
misc	*(Optional)* Total uncategorized charges in the payment. **Format:** Numeric; maximum 7 digits with optional decimal point and 2 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` • `Confirm` • `Cancel` • `Fraud` • `Other` **Example:** `"eventType":"Confirm"` Since 2.0
orderId	*(Optional)* Merchant's order ID associated with the payment. **Format:** Alphanumeric; maximum 100 characters Since 2.0
promoCode	*(Optional)* Promotion codes associated with the payment. Multiple promotion codes are separated by a period (.). **Format:** Alphanumeric; maximum 100 characters **Example:** `promoCode: "ABC"."DEF"."XYZ"` Since 2.0
reason	*(Optional)* Reason for the update. **Format:** Alphanumeric; maximum 255 characters Since 2.0

Update Payment Info Pixel Image Response

Update Payment Data Info Pixel Image Success 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 Pixel Image Examples

Update Payment Info Request

```
GET
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/updatepaymentinfo.gif
?token=x:1397855991:...
&apikey=...
&callId=...
&total=1...
&currencyCode=USD&
orderId=...
&promoCode=...
&reason=...
&subtotal=...
&shippingHandling=...
&tax=...
&discount=...
&giftWrap=...
&misc=...
```

Update Payment Info Error Response

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

SHA256–Bit Hashing A

SHA256–Bit Hashing Algorithm

SHA256-bit hashing is required for any string that includes your shared secret:

- The `x-pay-token` header in API calls.
- The `token` in the 1-pixel Update Payment Info image tag.

These cases are unrelated to the return or decryption of consumer payment information; specifically, SHA256-bit hashing is not used to decrypt payment data. The algorithms used for decryption are different.

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

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

SHA256–Bit Hashing Examples

SHA256–Bit Hash Java Example

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

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

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

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

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

SHA256–Bit Hash PHP Example

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

SHA256–Bit Hash Ruby Example

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

SHA256–Bit Hash Python Example

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

SHA256–Bit Hash C# Example

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

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

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

THIS PAGE INTENTIONALLY LEFT BLANK.

Clickjacking Prevention B

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.

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>
```

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.

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.

   Your test page should either be blank or display a message, such as **The content cannot be displayed in a frame.** If you can see the page that contains the Visa Checkout button, your prevention measures are not sufficient.

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.

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:

Java Servlet

The following sample implements a servlet to provide either the `X-FRAME-OPTIONS DENY` header or the `X-FRAME-OPTIONS SAMEORIGIN` header as a filter:

```
package com.your_package.filters;
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 mode = “DENY”;

    public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {
        HttpServletResponse res = (HttpServletResponse)response;
        chain.doFilter(request, response);
        res.addHeader(“X-FRAME-OPTIONS”, mode );
    }

    public void destroy() {
    }

    public void init(FilterConfig filterConfig) {
        String configMode = filterConfig.getInitParameter(“mode”);
        if ( configMode != null ) {
            mode = configMode;
        }
    }

}
```

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 id="WebApp_ID" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
    web-app_2_4.xsd">
    <display-name>ClickjackFilter</display-name>
    <filter>
        <filter-name>ClickjackFilterDeny</filter-name>
        <filter-class>com.merchant.filters.ClickjackFilter</filter-class>
        <init-param>
            <param-name>mode</param-name>
            <param-value>DENY</param-value>
        </init-param>
    </filter>

    <filter>
        <filter-name>ClickjackFilterSameOrigin</filter-name>
        <filter-class>com.merchant.filters.ClickjackFilter</filter-class>
        <init-param>
            <param-name>mode</param-name>
            <param-value>SAMEORIGIN</param-value>
        </init-param>
    </filter>

<!--
Use either the Deny or SameOrigin version. Do not use both versions at the same time.
-->

<!--
Use the Deny version to prevent everything, including your webapp, from framing the page.
-->
    <filter-mapping>
        <filter-name>ClickjackFilterDeny</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

<!--
Use SameOrigin to prevent everything, excepting your webapp, from framing the page.
-->
<!--
    <filter-mapping>
        <filter-name>ClickjackFilterSameOrigin</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
 -->

</web-app>
```

AVS and CVV Responses C

AVS Codes

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

AVS Code	Description
A	Partial match: street address matches, but 5-digit and 9-digit postal codes do not match.
B	Partial match: street address matches, but postal code is not verified. Returned only for non U.S.-issued Visa cards.
C	No match: street address and postal code do not match. Returned only for non U.S.-issued Visa cards.
D	Match: street address and postal code match. Returned only for non U.S.-issued Visa cards.
E	Invalid: AVS data is invalid or AVS is not allowed for this card type.
F	Partial match: card member's name does not match, but billing postal code matches. Returned only for the American Express card type.
G	Not supported: non-U.S. issuing bank does not support AVS.
H	Partial match: card member's name does not match, but street address and postal code match. Returned only for the American Express card type.
I	No match: address not verified. Returned only for non U.S.-issued Visa cards.
J	Match: card member's name, billing address, and postal code match. Shipping information verified and chargeback protection guaranteed through the Fraud Protection Program. Returned only if you are signed up to use AAV+ with the American Express Phoenix processor.
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.

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

CVV Codes

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

CVV Code	Description
M	The CVN matched.
P	The CVN was not processed by the processor for an unspecified reason.
S	The CVN is on the card but was not included in the request.
U	Card verification is not supported by the issuing bank.
X	Card verification is not supported by the card association.
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.

US, Canadian, and Australian Location Abbreviations

D

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

United States Abbreviations for States and Mailing Locations

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

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

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

Canadian Province Abbreviations

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

Australian State and Territory Abbreviations

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

State or Territory	Abbreviation
Tasmania	TAS
Northern Territory	NT

Document Revision History E

- 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

THIS PAGE INTENTIONALLY LEFT BLANK.