Collect.js

Overview

Collect.js is a JavaScript framework that allows merchants to collect sensitive payment information from their customers without exposing their website to the sensitive information. This can be done while allowing merchants to retain full control over the look and feel of their checkout experience.

This is a data collection and tokenization system, not a full payments API, so you can use this in conjunction with an existing transaction API (Direct Post) to submit transactions or use other gateway services that utilize payment information.

Usage

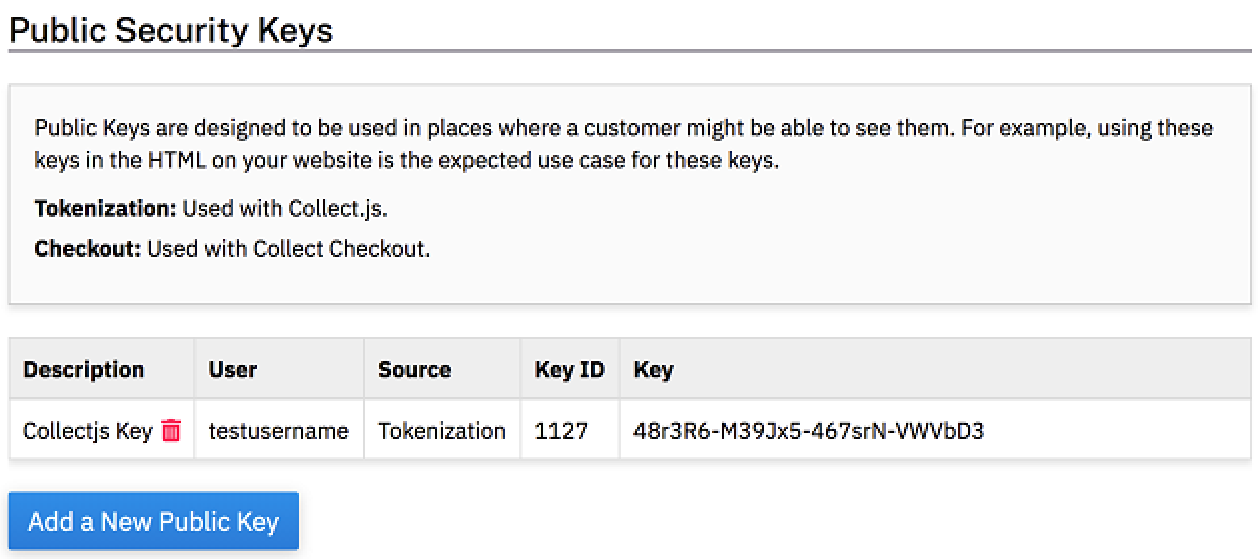
Collect.js is designed to be flexible, and its implementation can be as simple as pasting a single script tag to your checkout page, or it can be customized to interact with your website however you'd like.

Authentication

Authentication is done via a "tokenization key" that you can generate in your merchant control panel under the "Security Keys" settings page. Select a public key, and then "Tokenization" for the key permissions.

This tokenization key can only be used with Collect.js and will not work with any other APIs. Similarly, any API keys already created will not work with Collect.js.

This key will be visible to customers in your website's source code, so please make sure you only use the tokenization key here.



Public Security Keys

Public Keys are designed to be used in places where a customer might be able to see them. For example, using these keys in the HTML on your website is the expected use case for these keys.

Tokenization: Used with Collect.js.

Checkout: Used with Collect Checkout.

Description	User	Source	Key ID	Key
Collectjs Key	testusername	Tokenization	1127	48r3R6-M39Jx5-467srN-VWVbD3

The Payment Token

This is a new variable added to the Direct Post API that should be used in conjunction with this tool. This is what Collect.js will return to your website and it takes the place of the sensitive card or bank account information. It will look something like this:

```
3455zJms-7qA2K2-VdVrSu-Rv7WpvPuG7s8
```

This variable can be used in place of the existing ccnumber, ccexp, and cvv variables we have today. For ACH transactions (details below) it can be used in place of checkname, checkaba, and checkaccount.

The payment token can only be used once, and will expire after 24 hours if it is not used at all.

The payment token will also work when adding customers to the Customer Vault or recurring subscriptions. Just use "payment_token" where you were using the credit card and ACH account information before.

For example, if you would previously send this string:

```
type=sale
&amount=3.00&ccnumber=4111111111111111&ccexp=1020&cvv=123
```

Or:

```
type=sale&amount=3.00&checkname=Jane Doe
&checkaba=123123123&checkaccount=123123123
```

You could now send this:

```
type=sale&amount=3.00&payment_token=3455zJms-7qA2K2-VdVrSu-Rv7WpvPuG7s8
```

Test Tokens

If you would like to test using the payment token without using Collect.js to create one, you can use the below tokens to return test credit card and bank account information.

Payment Token Value	Test Data
00000000-000000-000000-000000000000	Card: 4111111111111111, Expiration: October 2025, CVV: 999
11111111-111111-111111-111111111111	ABA: 123123123, Account: 123123123, Name: Jane Doe

Integration Types

Collect.js supports two different ways to integrate with your site. Both offer the same basic functionality and security, so you can choose based on your interface and design requirements,

Lightbox Integration

The "lightbox" integration displays all sensitive payment fields in a single "pop-up" style display. All the entry and validation of payment data occurs within this single box; once valid information is provided, an event is provided for your page to capture the finished Payment Token.

Inline Integration

The "inline" integration allows you to seamlessly build Collect.js into your payment form. This solution allows you to create a payment form that looks and feels exactly like your website, but without the need for your service to handle any sensitive payment information.

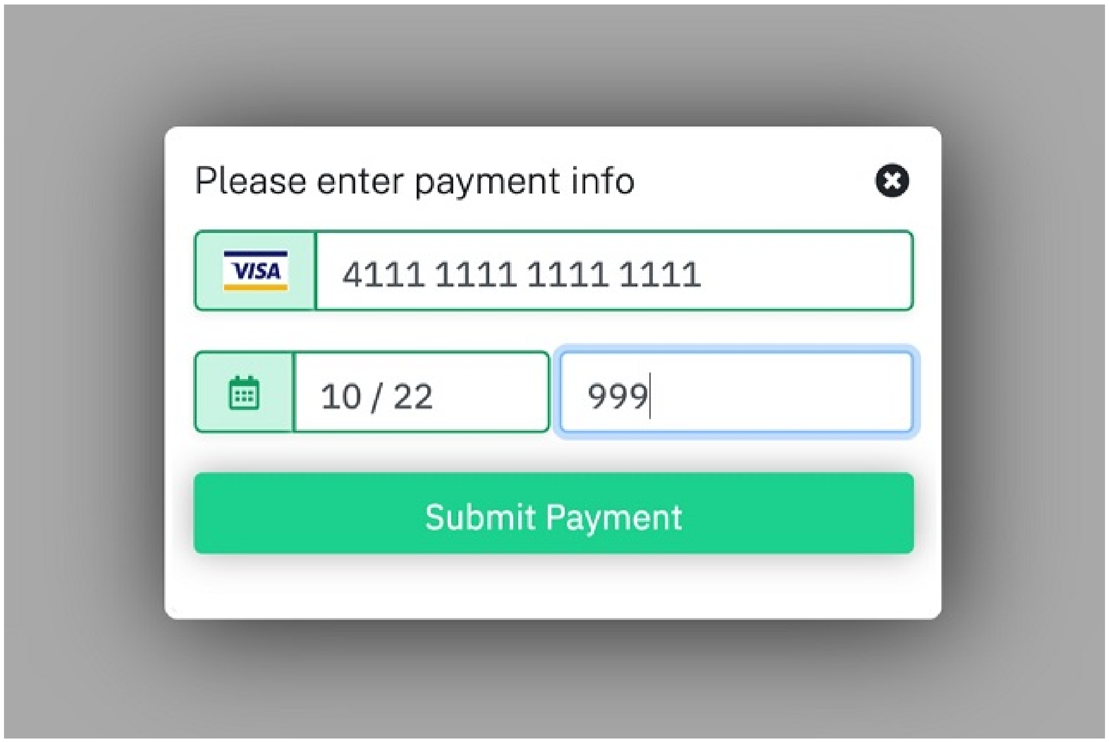
This works by creating iframes on your website for each credit card or electronic check field you need your customers to fill out. Using our custom "style sniffer" these fields will typically look exactly like the other fields on the page. If you want to just style them however you want, you can do that too by passing in custom CSS.

Simple Lightbox Implementation

The simplest way to integrate is by pasting in the following script tag to your web page (preferably in the header) where you'll be collecting payments:

```
<script src="https://centavo.transactiongateway.com/token/Collect.js"
data-tokenization-key="your-token-key-here"></script>
```

With this script, you just need to add a button with the ID of "payButton" to your page inside a form where you ask for the customer's information (name, address, email, etc.) You should make this button somewhere that indicates to the customer that they will be prompted to enter their card information and check out. Collect.js will find this button and display the below form in a lightbox over your website.



The customer will enter their card information and when they submit this mini-form, the lightbox will disappear, a hidden field will be inserted into your form with the "payment_token" value, and your form will be submitted.

You can then submit the transaction to the gateway with the Direct Post API using the "payment_token" variable.

Advanced Lightbox Implementation

If you want to have a little more control over the default behavior, you can pass in additional data elements in the script tag. Here’s an example using all the available variables:>

```
<script
src="https://centavo.transactiongateway.com/token/Collect.js" data-
tokenization-key="your-token-key-here" data-payment-
selector=".customPayButton"
data-primary-color="#ff288d"
data-theme="bootstrap"
data-secondary-color="#ffe200"
data-button-text="Submit the Payment" data-instruction-text="Enter
Card Information"
data-payment-type="cc"
data-field-cvv-display="hide"
data-price="1.00"
data-currency="USD"
data-country="US" data-field-google-pay-shipping-address-
required="true" data-field-google-pay-shipping-address-parameters-
phone-number-required="true" data-field-google-pay-shipping-address-
parameters-allowed-country-codes="US,CA" data-field-google-pay-billing-
address-required="true" data-field-google-pay-billing-address-
parameters-phone-number-required="true" data-field-google-pay-billing-
address-parameters-format="MIN" data-field-google-pay-email-
required="true"
></script>
```

Configuration Variables

Variable	Format	Behavior
data-tokenization-key	String	Authenticates the request
data-payment-selector	String	Tells Collect.js what class or id value will trigger the lightbox **Default: "#payButton"**
data-primary-color	String	The HEX value for the color of the submit button in the lightbox **Default: "#007BFF"**
data-theme	String ("bootstrap" or "material")	The version of the payment form customers will see. All available themes will use the primary and secondary colors provided. **Default: "bootstrap"**
data-secondary-color	String	The HEX value for the color of the lightbox border **Default: "#282828"**

Variable	Format	Behavior
data-button-text	String	The text that will display on the submit button in the lightbox **Default: "Submit Payment"**
data-instruction-text	String	The text that will display above the payment fields. Custom text should be short so as not to overlap with other elements in the lightbox. **Default: "Please enter payment info"**
data-payment-type	String ("cc" or "ck")	Whether the lightbox shows credit card or check fields ("cc" for credit cards or "ck" for checks) **Default: "cc"**
data-field-cvv-display	String ("show", "hide", or "required")	Whether the CVV field is required ("required"), optional ("show"), or not displayed at all ("hide"). Also supported as `data-field-cvv` for legacy users. **Default: "required"**
data-field-google-pay-selector	String	A CSS selector for the Google Pay field. **Default: "#googlepaybutton"**
data-field-google-pay-shipping-address-required	String ("true" or "false")	Determines whether or not Google Pay should capture shipping address information. Shipping information captured this way becomes stored in the payment token. **Default: "false"**
data-field-google-pay-shipping-address-parameters-phone-number-required	String ("true" or "false")	Determines whether or not Google Pay should capture a phone number from the user's shipping phone number. Phone numbers captured this way become stored in the payment token. **Default: "false"**
data-field-google-pay-shipping-address-parameters-allowed-country-codes	String (comma delimited list of 2 character country codes)	List of allowed countries. Credit cards from outside these countries will not be displayed as acceptable options within the Google Pay payment sheet. Omitting this value allows credit cards from any country. **Default: undefined**
data-field-google-pay-billing-address-required	String ("true" or "false")	Determines whether or not Google Pay should capture billing address information. Billing information captured this way becomes stored in the payment token. **Default: "false"**
data-field-google-pay-billing-address-parameters-phone-number-required	String ("true" or "false")	Determines whether or not Google Pay should capture a phone number from the user's billing phone number. Phone numbers captured this way become stored in the payment token. **Default: "false"**

Variable	Format	Behavior
data-field-google-pay-billing-address-parameters-format	String ("MIN" or "FULL")	Determines which billing address fields to capture from the user. "MIN" provides "zip", "country", "first_name" and "last_name". "FULL" additionally provides "address1", "address2", "city", "state". **Default: "MIN"**
data-field-google-pay-email-required	String ("true" or "false")	Determines whether or not Google Pay should capture an email address. Email addresses captured this way becomes stored in the payment token. **Default: "false"**
data-field-google-pay-button-type	String ("short" or "long")	"long" displays as "Buy with Google Pay". "short" displays the button with just the Google Pay logo. **Default: "long"**
data-price	String	The final cost that the user will be charged. **Default: undefined**
data-country	String	The country where the transaction is processed. **Required if using Google Pay**
data-currency	String	The currency the transaction will use to process the transaction. **Required if using Google Pay**

Collect.js Functions

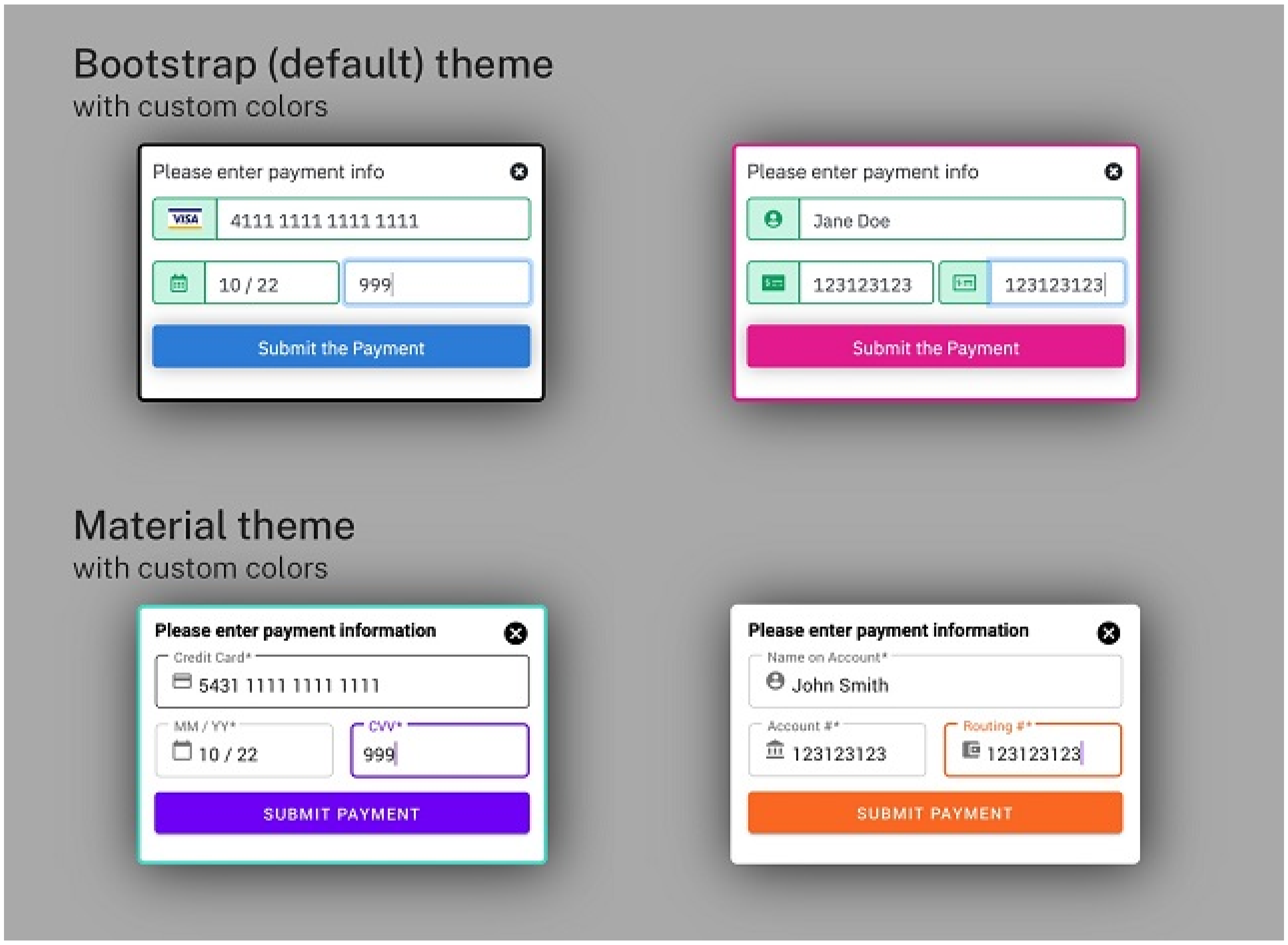
Function Name	Parameters	Description
configure	Object	Call this when you'd like to reconfigure Collect.js. Collect.js will try to run this automatically on page load, but you can run it manually to change the configuration at any time. This method optionally accepts an object with all configuration variables you're using for Collect.js.
startPaymentRequest	Event	Call this to bring up the lightbox with the secure payment form for the customer to fill out. If you are using the "payButton" ID or custom payment selector, this will automatically be called when the customer clicks that element on the page. This method accepts an event object as an optional parameter and will call the provided callback function with a token response and the optional event.

Function Name	Parameters	Description
closePaymentRequest		Call this to dismiss the lightbox. This replicates the behavior of the user clicking the "close" button inside the lightbox. No card or checking information will be saved.

You may also choose to configure Collect.js directly in your JavaScript, in which case you can do all of the above, and also implement a callback function that will execute when the customer submits the lightbox form. The payment token value will be returned in a "response" variable that you can do whatever you’d like with.

```
{
card: {
number: "411111******1111",
bin: "411111",
exp: "1028",
hash: "abcdefghijklmnopqrstuv1234567890",
},
check: {
name: null,
account: null,
hash: null,
aba: null,
},
token: "3455zJms-7qA2K2-VdVrSu-Rv7WpvPuG7s8",
initiatedBy: Event,
}
```

This implementation method allows for additional changes to the look and feel to better match your website's UI



Bootstrap (default) theme
with custom colors

Please enter payment info

4111 1111 1111 1111

10 / 22 | 999

Submit the Payment

Please enter payment info

Jane Doe

123123123 | 123123123

Submit the Payment

Material theme
with custom colors

Please enter payment information

Credit Card*: 5431 1111 1111 1111

MM / YY*: 10 / 22 | CVV*: 999

SUBMIT PAYMENT

Please enter payment information

Name on Account*: John Smith

Account #*: 123123123 | Routing #*: 123123123

SUBMIT PAYMENT

Expert Lightbox Implementation

If you have a webpage where you would like the lightbox to trigger without an element getting clicked, then you can call the following function:

```
CollectJS.startPaymentRequest(event)
```

This function will trigger the lightbox to show up and request payment details. If you wish to change any options, this should be done before calling this function since changes after this point wont affect the lightbox.

This function optionally receives an event object. If an event is passed into the startPaymentRequest function, that same event will exist in the callback's response variable under "response.initiatedBy". This can be used to track what event started the payment request and the next steps.

```
{
card: {
number: null,
bin: null,
exp: null,
hash: null,
},
check: {
name: "Jane Doe",
account: "1******23",
hash: "abcdefghijklmnopqrstuv1234567890",
aba: "123123123",
},
token: "3455zJms-7qA2K2-VdVrSu-Rv7WpvPuG7s8",
initiatedBy: Event,
}
```

If you wish to close the payment request without waiting for the user to click the close button, you can call the function:

```
CollectJS.closePaymentRequest()
```

This function will remove the lightbox from the page. No other functions will trigger from this function being called, including the callback.

Note that this implementation also requires you to include the standard script tag on the page as well.

Simple Integration Implementation

While the Inline integration model offers many customizable options, you can also get started quickly with a basic form. First, install the following JavaScript on your payment form page, preferably in the HEAD element:

```
<script src="https://centavo.transactiongateway.com/token/Collect.js"
data-tokenization-key="your-token-key-here" data-
variant="inline"></script>
```

This script assumes that you've set up a payment form already. The form can be laid out however you'd like, but there should be block-level elements (`div`, for example) where the sensitive payment info will be collected. The following IDs are expected to be used in place of standard form inputs:

For Credit Card Payments

- `ccnumber` (Credit card number)
- `ccexp` (Credit card expiration date)
- `cvv` (CVV)

For Electronic Check Payments

- `checkname` (Checking account name)
- `checkaccount` (Checking account number)
- `checkaba` (Routing number)

This is a very basic form that has integrated Inline Collect.js.

```
<form>
<input type="text" id="first_name">
<input type="text" id="last_name">
<input type="text" id="address">
<div id="ccnumber"></div>
<div id="ccexp"></div>
<div id="cvv"></div>
<input type="submit" id="payButton">
</form>
```

These elements will have iframes inserted into them, contents of which will be hosted by the gateway. They will be full width text fields and will use the style sniffer to match the rest of your page. The ID values let us know what field is collecting what information from the customer.

In addition to the empty fields, there must be a submit button in the form with an ID of " `payButton`." When the customer clicks this to submit the form, Collect.js will collect the data from all inline iframes and submit the form with a new "`payment_token`" value which is an encrypted version of the payment data.

After this form is submitted to your site, you can submit the data to the gateway via the Direct Post API. For example:

```
security_key: 3456h45k6b4k56h54kj6h34kj6445hj4
type: sale
amount: 4.00payment_token: 3455zJms-7qA2K2-VdVrSu-Rv7WpvPuG7s8
first_name: Jane
last_name: Doe
address: 123 Main St.
```

Advanced Implementation Method

If the simple implementation does not give you everything you need, then you can use the advanced implementation to customize the experience more to your liking. The options available are extensive, and you may use as many or as few as you want. Below is an example of using every variable possible.

```
<script
src="https://centavo.transactiongateway.com/token/Collect.js" data-
tokenization-key="your-token-key-here"
data-variant="inline"
data-payment-selector="#demoPayButton"
data-style-sniffer="false"
data-google-font="Montserrat:400" data-validation-callback =
"(function (field, valid, message) {console.log(field + ': ' + valid +
' -- ' + message)})"
data-custom-css='{
"background-color": "#a0a0ff",
"color": "#0000ff"
}'
data-invalid-css='{
"background-color":"red",
"color":"white"
}'
data-valid-css='{
"background-color":"#d0ffd0",
"color":"black"
}'
data-placeholder-css='{
"background-color":"#808080",
"color":"green"
}'
data-focus-css='{
"background-color":"#202020",
"color":"yellow"
}'
data-timeout-duration = "2000" data-timeout-callback = "(function()
{console.log('Timeout reached')})" data-fields-available-callback =
"(function() {console.log('Collect.js has added fields to the
form')})"
data-field-ccnumber-selector = '#demoCcnumber' data-field-ccnumber-
title = 'Card Number' data-field-ccnumber-placeholder = '0000 0000
0000 0000'
data-field-ccexp-selector = '#demoCcexp' data-field-ccexp-title =
'Expiration Date'
data-field-ccexp-placeholder = '00 / 00' data-field-cvv-display =
'required'
data-field-cvv-selector = '#demoCvv'
data-field-cvv-title = 'CVV Code'
data-field-cvv-placeholder = '***' data-field-checkaccount-selector =
'#demoCheckaccount'
data-field-checkaccount-title = 'Account Number' data-field-
checkaccount-placeholder = '000000000000'
```

```
data-field-checkaba-selector
= '#demoCheckaba'
data-field-checkaba-title = 'Routing Number' data-field-checkaba-
placeholder = '000000000' data-field-checkname-selector =
'#demoCheckname'
data-field-checkname-title = 'Account Name' data-field-checkname-
placeholder = 'Customer Name'
data-price="1.00"
data-currency="USD"
data-country="US" data-field-google-pay-shipping-address-
required="true" data-field-google-pay-shipping-address-parameters-
phone-number-required="true" data-field-google-pay-shipping-address-
parameters-allowed-country-codes="US,CA" data-field-google-pay-billing-
address-required="true" data-field-google-pay-billing-address-
parameters-phone-number-required="true" data-field-google-pay-billing-
address-parameters-format="MIN" data-field-google-pay-email-
required="true"
></script>
```

Configuration Variables

Variable	Format	Behavior
data-tokenization-key	String	Authenticates the request
data-variant	String ("inline" or "lightbox")	Whether to use "inline" or "lightbox" integration **(required for inline integration)** **Default: "lightbox"**
data-payment-selector	String	Tells Collect.js what class or id value will trigger the form submission **Default: "#payButton"**
data-style-sniffer	String ("true" or "false")	Whether Collect.js should try to calculate the style of form fields in your current form and use that as a baseline style for the Collect.js fields ("true" to calculate style, "false" to start with unstyled text fields) **Default: "true"**
data-validation-callback	String	A JavaScript function which will be called each time a Collect.js field attempts to validate. It will recieve three paramaters: a string indicating which field was validated (ccnum or checkname, for example), a boolean for whether or not it validated successfully, and a string which may provide more detailed information about why the validation failed. For broadest compatibility, enclose the function in parentheses like in the example above
data-custom-css	JSON String	The CSS rules that will be applied to the fields by default. These override anything provided through the style-sniffer, if used. The rules should be packaged as a JSON-formatted object, containing a key-value pair for each property's name and value. Please see below for a list of the supported CSS properties

Variable	Format	Behavior
data-invalid-css	JSON String	The CSS rules that will be added to a field when it fails to validate. These override anything provided through the style-sniffer and the custom-css paramater, if used. The rules should be packaged as a JSON-formatted object, containing a key-value pair for each property's name and value. Please see below for a list of the supported CSS properties
data-placeholder-css	JSON String	The CSS rules that will be added to a field when it's displaying a placeholder. The rules should be packaged as a JSON-formatted object, containing a key-value pair for each property's name and value. Please see below for a list of the supported CSS properties
data-focus-css	JSON String	The CSS rules that will be added to a field when it has the keyboard focus. The rules should be packaged as a JSON-formatted object, containing a key-value pair for each property's name and value. Please see below for a list of the supported CSS properties
data-valid-css	JSON String	The CSS rules that will be added to a field when it successfully validates and saves. These override anything provided through the style-sniffer and the custom-css paramater, if used. The rules should be packaged as a JSON-formatted object, containing a key-value pair for each property's name and value. Please see below for a list of the supported CSS properties
data-google-font	String	Directs Collect.js to load font collections available through Google Fonts. This only makes the fonts available in the fields; you must still provide (either directly or through the style sniffer) styles that specify them. List the font name, followed by a colon and the specific weights or variants needed. **Example: "Open Sans:400,700i"**
data-timeout-duration	Integer	When form submission is triggered, Collect.js will wait only this long (in milliseconds) for payment data validation and recording to complete. If, by this time, Collect.js is still missing confirmation on vital fields, the `data-timeout-callback` function will be invoked **Default: "0" which disables the timeout**

Variable	Format	Behavior
data-timeout-callback	String	A JavaScript function which gets called if `data-timeout-duration` has passed since we tried to submit the form, but we still haven't confirmed that enough fields are stored with the token to make a viable payment. This allows for the site to retry submission, or ask the customer to try submission again, if an invalid entry or intermittent connection caused the data storage to fail. For broadest compatibility, enclose the function in parentheses like in the example above **Default: an internal function that displays a "Please submit the form again." alert**
data-fields-available-callback	String	A JavaScript function which gets called once Collect.js has installed the fields onto your page. A typical use case is to wire up event handlers to the fields when they are enterred or left. For broadest compatibility, enclose the function in parentheses like in the example above
data-field-ccnumber-selector	String (CSS Selector)	A CSS selector for the Credit Card Number inline field **Default: "#ccnumber"**
data-field-ccnumber-title	String	A title for the Credit Card Number inline field
data-field-ccnumber-placeholder	String	Placeholder text for the Credit Card Number inline field
data-field-ccexp-selector	String (CSS Selector)	A CSS selector for the Credit Card Expiration Date inline field **Default: "#ccexp"**
data-field-ccexp-title	String	A title for the Credit Card Expiration Date inline field
data-field-ccexp-placeholder	String	Placeholder text for the Credit Card Expiration Date inline field
data-field-cvv-display	String ("show", "hide", or "required")	Whether the CVV field is required ("required"), optional ("show"), or not displayed at all ("hide"). If the CVV field is required, a space for it must be provided on the form. Also supported as `data-field-cvv` for legacy users **Default: "required"**
data-field-cvv-selector	String (CSS Selector)	A CSS selector for the CVV inline field **Default: "#cvv"**
data-field-cvv-title	String	A title for the CVV inline field
data-field-cvv-placeholder	String	Placeholder text for the CVV inline field

Variable	Format	Behavior
data-field-checkaccount-selector	String (CSS Selector)	A CSS selector for Checking Account Number inline field **Default: "#checkaccount"**
data-field-checkaccount-title	String	A title for the Checking Account Number inline field
data-field-checkaccount-placeholder	String	Placeholder text for the Checking Account Number inline field
data-field-checkaba-selector	String (CSS Selector)	A CSS selector for the Checking Routing Number inline field **Default: "#checkaba"**
data-field-checkaba-title	String	A title for the Checking Routing Number inline field
data-field-checkaba-placeholder	String	Placeholder text for the Checking Routing Number inline field
data-field-checkname-selector	String (CSS Selector)	A CSS selector for the Checking Account Name inline field **Default: "#checkname"**
data-field-checkname-title	String	A title for the Checking Account Name inline field
data-field-checkname-placeholder	String	Placeholder text for the Checking Account Name inline field
data-field-google-pay-selector	String	A CSS selector for the Google Pay field. **Default: "#googlepaybutton"**
data-field-google-pay-shipping-address-required	String ("true" or "false")	Determines whether or not Google Pay should capture shipping address information. Shipping information captured this way becomes stored in the payment token. **Default: "false"**
data-field-google-pay-shipping-address-parameters-phone-number-required	String ("true" or "false")	Determines whether or not Google Pay should capture a phone number from the user’s shipping phone number. Phone numbers captured this way become stored in the payment token. **Default: "false"**
data-field-google-pay-shipping-address-parameters-allowed-country-codes	String (comma delimited list of 2 character country codes)	List of allowed countries. Credit cards from outside these countries will not be displayed as acceptable options within the Google Pay payment sheet. Omitting this value allows credit cards from any country. **Default: undefined**

Variable	Format	Behavior
data-field-google-pay-billing-address-required	String ("true" or "false")	Determines whether or not Google Pay should capture billing address information. Billing information captured this way becomes stored in the payment token. **Default: "false"**
data-field-google-pay-billing-address-parameters-phone-number-required	String ("true" or "false")	Determines whether or not Google Pay should capture a phone number from the user's billing phone number. Phone numbers captured this way become stored in the payment token. **Default: "false"**
data-field-google-pay-billing-address-parameters-format	String ("MIN" or "FULL")	Determines which billing address fields to capture from the user. "MIN" provides "zip", "country", "first_name" and "last_name". "FULL" additionally provides "address1", "address2", "city", "state". **Default: "MIN"**
data-field-google-pay-email-required	String ("true" or "false")	Determines whether or not Google Pay should capture an email address. Email addresses captured this way becomes stored in the payment token. **Default: "false"**
data-field-google-pay-button-type	String ("short" or "long")	"long" displays as "Buy with Google Pay". "short" displays the button with just the Google Pay logo. **Default: "long"**
data-price	String	The final cost that the user will be charged. **Default: undefined**
data-country	String	The country where the transaction is processed. **Required if using Google Pay**
data-currency	String	The currency the transaction will use to process the transaction. **Required if using Google Pay**

Collect.js Functions

Function Name	Parameters	Description
configure	Object	Call this when you'd like to reconfigure Collect.js. Collect.js will try to run this automatically on page load, but you can run it manually to change the configuration at any time. This will draw or re-draw all iframes onto the page. This method optionally accepts an object with all configuration variables you're using for Collect.js.

Function Name	Parameters	Description
startPaymentRequest	Event	Call this when you want to save the data in the iframes and get the token value in the callback. This method accepts an event object as an optional parameter. It will call the provided callback function with a token response and the optional event.
clearInputs		Call this when you want to clear whatever the user has entered into any input provided by Collect.js.

JavaScript Based Activation

You may also choose to configure Collect.js directly in your JavaScript, For this, you will typically only include the `data-tokenization-key` parameter in the script tag, and deploy the other options with a `CollectJS.configure()` call. See the Advanced Inline JavaScript Example for a demonstration with the main available options.

The `CollectJS.configure` function also lets you specify a `callback` function that will execute when the customer submits the payment form and payment info has been successfully stored. The callback takes the place of the default "add the payment token and submit the form" behavior and gets passed a "response" variable with the Payment Token. It is your responsibility to ensure this is posted to your server.

```
{
card: {
number: "411111******1111",
bin: "411111",
exp: "1028",
hash: "abcdefghijklmnopqrstuv1234567890",
},
check: {
name: null,
account: null,
hash: null,
aba: null,
},
token: "3455zJms-7qA2K2-VdVrSu-Rv7WpvPuG7s8",
initiatedBy: Event,
}
```

Styling Limitations

For security and compatibility reasons, the styling system- whether provided via `custom-css`, `invalid-css`, `valid-css`, `focus-css` or calculated using the style-sniffer, only supports the following CSS properties:

- background-color
- border-bottom-color

- border-bottom-left-radius
- border-bottom-right-radius
- border-bottom-style
- border-bottom-width
- border-left-color
- border-left-style
- border-left-width
- border-right-color
- border-right-style
- border-right-width
- border-top-color
- border-top-left-radius
- border-top-right-radius
- border-top-style
- border-top-width
- border-width
- border-style
- border-radius
- border-color
- bottom
- box-shadow
- color
- cursor
- direction
- font-family
- font-kerning
- font-size
- font-stretch
- font-style
- font-variant-caps
- font-variant-numeric
- font-weight
- height
- letter-spacing
- line-height
- margin-top
- margin-bottom
- opacity
- outline-color
- outline-offset
- outline-style
- outline-width
- padding
- padding-bottom
- padding-left
- padding-right
- padding-top

- pointer-events
- text-align
- text-align-last
- text-decoration
- text-decoration-line
- text-decoration-style
- text-decoration-color
- text-decoration-skip-ink
- text-underline-position
- text-indent
- text-rendering
- text-shadow
- text-size-adjust
- text-overflow
- text-transform
- transition
- vertical-align
- white-space
- will-change
- word-break
- word-spacing
- hyphens

Placeholder CSS can only use the following attributes

- background-color
- font-family
- font-kerning
- font-size
- font-stretch
- font-style
- font-variant-caps
- font-variant-numeric
- font-weight
- word-spacing
- letter-spacing
- line-height
- text-decoration
- text-indent
- text-transform
- transition
- vertical-align
- opacity
- color

Any other CSS properties will be ignored.

Expert Inline Implementation

Understanding the lifecycle of a Collect.js-enabled form

Each field that Collect.js supplies communicates with your Gateway independently. These fields will check the payment information, and if valid, direct the Gateway to save it, as soon as the customer exits a field. This process triggers the validation callbacks you can use to monitor the user's progress and control their interactions with your form.

When the submit button is pressed (or `CollectJS.startPaymentRequest` is manually called), Collect.js directs each field to validate and save one last time. Once it gets back a notice of successful validation and saving from enough fields to make a viable payment request, it proceeds to submit the form or call an alternative `callback` as configured.

Once the Payment Token is used in a Direct Post request, it's automatically destroyed. This prevents its reuse for a later unauthorized charge, but means that if you have to collect the payment information again (for example, after a declined transaction), you're going to have to start fresh and generate a new token.

Manually Triggering Payment Information Saving

You can take control of when the final validate-and-save process is triggered. Rather than binding it explicitly to a payment button, you can call the following JavaScript function when ready.

```
CollectJS.startPaymentRequest(event)
```

When triggered, this causes the same behavior as pressing a payment button does by default: all the fields are told to validate and save. Once we confirm all data is stored, the callback you configured is executed.

This function optionally receives an event object. If an event is passed into the startPaymentRequest function, that same event will exist in the callback's response variable under "response.initiatedBy". This can be used to track what event started the recording request and the next steps.

```
{
card: {
number: null,
bin: null,
exp: null,
hash: null,
},
check: {
name: "Jane Doe",
account: "1******23",
hash: "abcdefghijklmnopqrstuv1234567890",
aba: "123123123",
},
token: "3455zJms-7qA2K2-VdVrSu-Rv7WpvPuG7s8",
initiatedBy: Event,
}
```

Note that this implementation also requires you to include the standard script tag on the page as well.

Integration with form validation

While Collect.js doesn't let you directly access the contents of the payment information fields, it does provide several ways to check if they contain *valid* content. There are two distinct ways you can access this information in your form validation code:

1. Supply a "validation-callback"

This will listen for all Collect.js validation changes. This function will get a notice about each field change, and you can keep a tally during the form's life.

```
<script
src="https://centavo.transactiongateway.com/token/Collect.js" data-
tokenization-key="your-token-key-here"
data-validation-callback="(
function(fieldName, valid, message) {
if (valid) {
... store the fact that fieldName is valid ...
} else { ... remove fieldName from the valid list, maybe display
message to the user ...
}
}
)">
```

2. Check CSS classes.

When you start the validation process, you can see if the elements you loaded Collect.js fields into have either `CollectJSValid` or `CollectJSInvalid` elements within them. Note that blank fields, or some fields in the process or being edited or saved, will have neither set. You can decide how to handle these depending on when you're performing the check, what field is blank or unsaved, and how that fits into your site's flow.

```
validCardNumber = document.querySelector("#ccnumber .CollectJSValid")
!== null;validExpiration = document.querySelector("#ccexp
.CollectJSValid") !== null;validCvv = document.querySelector("#cvv
.CollectJSValid") !== null;invalidCvv = document.querySelector("#cvv
.CollectJSInvalid") !== null;blankOrUnsavedCvv = !validCvv &&
!invalidCvv;
```

Blur and Focus Events

Some styling techniques will change the classes of related elements as a user enters and leaves a form field. Google's Material Design Components for the Web is a typical example-- the label moves above the text, and an underline that's not part of the field changes color. Collect.js exposes `focus` and `blur` events that can be used to trigger these types of effects with Collect.js fields.

Here's a tangible example. The `data-fields-available-callback` code adds a listener to each Collect.js field's `blur` and `focus` events. This listener adds or removes the `active` class from the nearest label. When a user enters the field, the label next to it changes from gray to bold and blue, reverting once they leave the field.

```
<script
src="https://centavo.transactiongateway.com/token/Collect.js" data-
tokenization-key="your-token-key-here"
data-variant="inline"
data-fields-available-callback='
(function() { var frames = document.querySelectorAll(".input-field
iframe.CollectJSInlineIframe")
for (var i = 0; i
```

When the `blur` event is fired, it will include a `detail` structure with one element: `empty`. This tells you if the field is blank, so you can style it differently, without disclosing its contents.

Google Pay

Image not found or type unknown

Google Pay allows customers to provide credit card data saved in their Google accounts to be used in online payments. Collect.js supports Google Pay in both lightbox and inline integrations allowing you to capture these credit card details in either flow. And to make the integration as seamless as possible, the Google Pay data will be returned to you in the "payment_token" variable, so no matter what payment method your customers make, your transaction request can be exactly the same.

To use Google Pay, you must provide Collect.js country and currency values. These values are used to ensure the user can only select a valid card. You must also provide an HTML element on your page that Collect.js can use to draw the Google Pay button.

Google Pay is currently supported on TSYS - EMV and Elavon viaConex.

```
<html>
<head>
<script
src="https://centavo.transactiongateway.com/token/Collect.js" data-
tokenization-key="000000-000000-000000-000000"
data-variant="inline"
data-country="US"
data-price="1.00"
data-currency="USD"
></script>
</head>
<body>
<form action="submit_to_direct_post_api.php" method="post"> <div
id="googlepaybutton"></div>
</form>
</body></html>
```

This will create a Google Pay button that will be inserted in the div as an iframe. The Google Pay button will be 240x70px, so make sure to leave room in this div for the button to display in full. When a user clicks the button, they are presented with a payment sheet requesting the user’s payment details. After the user submits the payment sheet, Collect.js executes the callback function if one were provided, or submits your form with the payment token attached.

Capture Billing and Shipping Data

Collect.js also allows you to capture the user’s shipping and billing details with Google Pay, just like the credit card data. This eliminates the need for you to capture this manually in your own web form. When these options are enabled, Google Pay will also request the user’s information and Collect.js will store all that data in the payment token.

In addition to the payment data that gets stored for all Google Pay transactions, the payment token will include the following shipping fields when shipping address required is enabled:

- shipping_address_1
- shipping_address_2
- shipping_zip
- shipping_city
- shipping_state
- shipping_country
- shipping_firstname
- shipping_lastname
- phone (also requires phone_number_required to be enabled)

When billing_address_required is enabled, Collect.js will also capture these fields:

- address1 (also requires format to be "FULL")
- address2 (also requires format to be "FULL")
- zip
- city (also requires format to be "FULL")
- state (also requires format to be "FULL")
- country
- firstname
- lastname
- phone (also requires phone_number_required to be enabled)

```
<html>
<head>
<script
src="https://centavo.transactiongateway.com/token/Collect.js" data-
tokenization-key="000000-000000-000000-000000"
data-variant="inline" data-field-google-pay-selector=".google-pay-
button"
data-field-google-pay-shipping-address-required="true" data-field-
google-pay-shipping-address-parameters-phone-number-required="true"
data-field-google-pay-shipping-address-parameters-allowed-country-
codes="US,CA"
data-field-google-pay-billing-address-required="true" data-field-
google-pay-billing-address-parameters-phone-number-required="true"
data-field-google-pay-billing-address-parameters-format="MIN"
></script>
</head>
<body>
<form action="submit_to_direct_post_api.php" method="post"> <div
class="google-pay-button"></div>
</form>
</body>
```

```
</html>
```