Use the Clover iframe to create a payment form

North America—United States and Canada

The Clover in-line frame (iframe) lets you insert an HTML page into another HTML-based webpage, such as your merchant's website. With the Clover iframe integration, your ecommerce website can communicate with the Clover Ecommerce APIs. You can build a secure payment experience on your website using iframe fields and elements to get the customer's payment information securely from the merchant browser to the Clover server.

Prequisite: Browser support

Apple Safari, Google Chrome, Mozilla Firefox, and Microsoft Edge browsers support the Clover iframe.

> ❗ **Important**
>
> **Secure your site**—Developers using the Clover iframe integration are advised to check their code and remove any reference to `cdn.polyfill.io` . For example:

```
<head>
...
delete this→ <script src= "https://cdn.polyfill.io/v3/polyfill.min.js"> </script> ←
<script src= "https://checkout.sandbox.dev.clover.com/sdk.js"> </script>
</head>
```

For additional information about the security concern with using polyfill script, see the related announcement.

Step 1: Add the Clover SDK to your webpage

1. To use the Clover iframe features, you need to import the Clover SDK to your webpage.
   Add a `<script>` block to the `<head>` block of your webpage.
2. In your HTML form, add `<script>` to import the Clover SDK and use the Clover iframe features. Use the sandbox or production `sdk.js` file to define the source (src).
   Sandbox: `<https://checkout.sandbox.dev.clover.com/sdk.js>`
   Production: `<https://checkout.clover.com/sdk.js>`

HTML

```
<head>
  ...
  <script src="https://checkout.sandbox.dev.clover.com/sdk.js"></script>
</head>
```

Step 2: Configure the SDK

To make payment requests on a merchant's behalf, set up the Clover SDK to use the merchant's public key retrieved from the PAKMS endpoint.

1. Create a `clover` constant (`const`) and pass the merchant's key as the parameter of a new `Clover` object.
2. Create another constant (`const`) for `clover.elements()` . This constant creates card tokens based on information entered in the iframe. Each entity in the iframe is an element.

JavaScript

```
merchantId to const clover.
const clover = new Clover('12a3b456789c12345d67891234e56f78', {
    merchantId: 'xxxxxxxxxxxxx'
});
const elements = clover.elements();
```

Note: merchantId is required.

3. If you want to configure a language, add a `locale` to `const clover` . Clover supports:

- English USA (`en-US`) by default
- English Canadian (`en-CA`)
- French Canadian (`fr-CA`). For example, the following appended command supports French Canadian:

JavaScript

```
const clover = new Clover('12a3b456789c12345d67891234e56f78', {
    locale: 'fr-CA'
});
const elements = clover.elements();
```

4. To support the reCAPTCHA verification service, street address input field, or other optional features, add `merchantId` to the iframe configuration.

JavaScript

```
merchantId to const clover.
const clover = new Clover('12a3b456789c12345d67891234e56f78', {
    merchantId: 'xxxxxxxxxxxxx'
});
const elements = clover.elements();
```

> 🚧 **IMPORTANT**
>
> Apps for a single merchant can hardcode the public key to initialize the iframe SDK.
>
> If your app uses a model where a customer may expect to use one card token between multiple merchants, send an email to Clover developers relations team.

Step 3: Set up the payment form

You can create an HTML `<form>` where customers enter their credit card information. To set up your payment form:

1. Add a `<form>` to contain card data fields on your webpage.
2. Set the `id` attribute and make a note of this value. Example: `payment-form` .

HTML

```
<body>

  <form action="/charge" method="post" id="payment-form">
    <!-- this form contains the card data fields -->
  </form>

</body>
```

3. In the `<form>` , create an `<input>` field to enter the amount of the charge.

HTML

```
<form action="/charge" method="post" id="payment-form">

  <div class="form-row top-row">
    <div id="amount" class="field card-number">
      <input name="amount" placeholder="Amount">
    </div>
  </div>

</form>
```

4. Add `<div>` containers to enable customers to enter their card details. For each card data field, add a `<div>` to display error messages (`class="input-errors"`).

HTML

```
<form action="/charge" method="post" id="payment-form">
  ...

  <div class="form-row top-row">
    <div id="amount" class="field card-number">
      <input name="amount" placeholder="Amount">
    </div>
  </div>

  <div class="form-row top-row">
    <div id="card-number" class="field card-number"></div>
    <div class="input-errors" id="card-number-errors" role="alert"></div>
  </div>

  <div class="form-row">
    <div id="card-date" class="field third-width"></div>
    <div class="input-errors" id="card-date-errors" role="alert"></div>
  </div>

  <div class="form-row">
    <div id="card-cvv" class="field third-width"></div>
    <div class="input-errors" id="card-cvv-errors" role="alert"></div>
  </div>

  <div class="form-row">
    <div id="card-postal-code" class="field third-width"></div>
    <div class="input-errors" id="card-postal-code-errors" role="alert"></div>
  </div>

  <div id="card-response" role="alert"></div>

  ...
</form>
```

5. Add a `<button>` for the users to finalize their payment.

HTML

```
<form action="/charge" method="post" id="payment-form">
  ...
```

```
  <div class="button-container">
    <button>Submit Payment</button>
  </div>

</form>
```

After the payment `<form>` is set up, you can use JavaScript components to make it interactive. See Create an interactive payment.

Step 4: Create an interactive payment

To make the payment `<form>` interactive, add JavaScript components provided with the iframe.

1. Complete the steps to set up the payment form.
2. Use the `<form>` element ID from the set up the payment form section to create a constant to access the payment form.

JavaScript

```
const form = document.getElementById('payment-form');
```

3. Create instances of the card elements and mount them to the `<div>` containers. When creating containers, you can add CSS styling as the second parameter to match your website branding. See Customize iframe elements with CSS.

JavaScript

```
const cardNumber = elements.create('CARD_NUMBER', styles);
const cardDate = elements.create('CARD_DATE', styles);
const cardCvv = elements.create('CARD_CVV', styles);
const cardPostalCode = elements.create('CARD_POSTAL_CODE', styles);

cardNumber.mount('#card-number');
cardDate.mount('#card-date');
cardCvv.mount('#card-cvv');
cardPostalCode.mount('#card-postal-code');
```

4. Add event listeners (`addEventListener`) for displaying any error messages to the user.

The SDK's real-time validation of the card data fields ensures that the customer entries match the expected format. With the change and blur event listeners, you can handle real-time validation in the iframe.

JavaScript Sample iframe validation response

```
const cardResponse = document.getElementById('card-response');
const displayCardNumberError = document.getElementById('card-number-errors');
const displayCardDateError = document.getElementById('card-date-errors');
const displayCardCvvError = document.getElementById('card-cvv-errors');
const displayCardPostalCodeError = document.getElementById('card-postal-code-errors');

  // Handle real-time validation errors from the card element
  cardNumber.addEventListener('change', function(event) {
    console.log(`cardNumber changed ${JSON.stringify(event)}`);
  });

  cardNumber.addEventListener('blur', function(event) {
    console.log(`cardNumber blur ${JSON.stringify(event)}`);
  });

  cardDate.addEventListener('change', function(event) {
    console.log(`cardDate changed ${JSON.stringify(event)}`);
  });

  cardDate.addEventListener('blur', function(event) {
    console.log(`cardDate blur ${JSON.stringify(event)}`);
  });

  cardCvv.addEventListener('change', function(event) {
    console.log(`cardCvv changed ${JSON.stringify(event)}`);
  });

  cardCvv.addEventListener('blur', function(event) {
    console.log(`cardCvv blur ${JSON.stringify(event)}`);
  });

  cardPostalCode.addEventListener('change', function(event) {
    console.log(`cardPostalCode changed ${JSON.stringify(event)}`);
  });

  cardPostalCode.addEventListener('blur', function(event) {
    console.log(`cardPostalCode blur ${JSON.stringify(event)}`);
  });
```

5. Add an event listener (`addEventListener`) to the submit event. This listener takes the validated card data from the payment form and calls the `clover.createToken()` method. A token is generated.

JavaScript

```
// Listen for form submission
form.addEventListener('submit', function(event) {
  event.preventDefault();
  // Use the iframe's tokenization method with the user-entered card details
  clover.createToken()
    .then(function(result) {
    if (result.errors) {
      Object.values(result.errors).forEach(function (value) {
        displayError.textContent = value;
      });
    } else {
      cloverTokenHandler(result.token);
    }
  });
});
```

Sample tokenization result

```
{
        "token": "clv_1TST39I92..."
}
```

6. Add the generated token to the server application to charge the tokenized card.

JavaScript

```
function cloverTokenHandler(token) {
  // Insert the token ID into the form so it gets submitted to the server
  var form = document.getElementById('payment-form');
  var hiddenInput = document.createElement('input');
  hiddenInput.setAttribute('type', 'hidden');
  hiddenInput.setAttribute('name', 'cloverToken');
  hiddenInput.setAttribute('value', token);
  form.appendChild(hiddenInput);
  form.submit();
}
```

Step 5: Create a charge

Once you create an interactive payment form and generate a card token (see Generate a card token) you can now create a charge. For detailed information, see Create a charge.

Prerequisites

- Create a charge request must be a server-to-server call as shown in the information flow for a charge request.
- Use an `idempotency` key, which is a required value generated by your app for Clover, to safely retry the `v1/charges` request without accidental double charges. See Use idempotency keys for more information.

Steps

1. On the payment form, enter the customer's card details to pay for an order.
2. Send the `amount` and the token (as the value of `source`) to the `/v1/charges` endpoint to complete the transaction.
3. Set the `authorization: Bearer` as your OAuth-generated `auth_token` . The charge is processed for the specified amount using the token.

cURL

```
curl --request POST \
  --url 'https://scl-sandbox.dev.clover.com/v1/charges' \
  --header 'accept: application/json' \
```

```
--header 'authorization: Bearer {access_token}' \
--header 'idempotency-key {uuid4_key}' \
--header 'content-type: application/json' \
--header 'x-forwarded-for: {client_ip}' \
--data '{"amount":4500,"currency":"usd","source":"clv_1ABCD7234efghDIJKlMNOp5qrS"}'
```

Related topics

- Clover iframe integrations
- Clover iframe—Card and page elements
- Customize iframe elements with CSS

Updated 2 minutes ago

← Clover iframe integrations

Clover iframe—Card and page elements →