

## Stripe.js reference

This reference documents every object and method available in Stripe's browser-side JavaScript library, Stripe.js. Use our [React Stripe.js reference](#) if you want to add Elements to your React based app.

You can use Stripe.js' APIs to tokenize customer information, collect sensitive payment details using customizable [Stripe Elements](#), and accept payments with [browser payment APIs](#) like Apple Pay and the Payment Request API.

---

## Including Stripe.js

Include the Stripe.js script on each page of your site—it should always be loaded directly from <https://js.stripe.com>, rather than included in a bundle or hosted yourself.

To best leverage Stripe's advanced fraud functionality, include this script on every page, not just the checkout page. This [allows Stripe to detect suspicious behavior](#) that may be indicative of fraud as customers browse your website.

### Using Stripe.js as a module

We also provide an npm package that makes it easier to load and use Stripe.js as a module. For more information, check out the [project on GitHub](#).

## Asynchronous and deferred loading of Stripe.js

Asynchronous loading of JavaScript is generally recommended, as it can improve the user experience of your site by not blocking DOM rendering during [script loading](#).

The easiest way to asynchronously load Stripe.js is to use the npm module as described above. It does asynchronous loading by default.

You can also load Stripe.js using the `async` or `defer` attribute on the script tag.

Note, however, that with asynchronous loading any API calls will have to be made only after the script execution has finished.

---

### `Stripe(publishableKey, options?)`

Use `Stripe(publishableKey, options?)` to create an instance of the **Stripe object**. The Stripe object is your entrypoint to the rest of the Stripe.js SDK.

Your Stripe publishable [API key](#) is required when calling this function, as it identifies your website to Stripe.

We've prefilled the example with a sample [test API key](#). Don't submit any personally identifiable information in requests made with this key. To create a Stripe object using your account, replace the sample API key with your actual API key or [sign in](#).

When you're ready to accept live payments, replace the test key with your live key in production. Learn more about how API keys work in [test mode and live mode](#).

## Method parameters

**publishableKey** REQUIRED string

Your publishable key.

**options** optional object

Initialization options.

+ Show options properties

## The Elements object

Stripe Elements are customizable UI components used to collect sensitive information in your payment forms.

Use an `Elements` instance to create and manage a group of individual `Element` instances.

`stripe.elements(options?)`

This method creates an `Elements` instance, which manages a group of elements.

## Method parameters

**options** optional object

A set of options to create this `Elements` instance with.

[× Hide options properties](#)**fonts** array

An array of custom fonts, which elements created from the `Elements` object can use. Fonts can be specified as [CssFontSource](#) or [CustomFontSource](#) objects.

**locale** string

A **locale** to display placeholders and error strings in. Default is `auto` (Stripe detects the locale of the browser).

Setting the locale does not affect the behavior of postal code validation—a valid postal code for the billing country of the card is still required.

**clientSecret** CONDITIONALLY REQUIRED string

Required to use with the [Payment Element](#) and the [Link Authentication Element](#).

The **client secret** for a PaymentIntent or SetupIntent.

**appearance** object

Supported for the [Payment Element](#), the [Link Authentication Element](#), and the [Address Element](#).

Match the design of your site with the **appearance option**. The layout of each Element stays consistent, but you can modify colors, fonts, borders, padding, and more.

**loader** 'auto' | 'always' | 'never'

Supported for the [Payment Element](#), the [Link Authentication Element](#), and the [Address Element](#).

Display skeleton loader UI while waiting for Elements to be fully loaded, after they are mounted. Default is `'auto'` (Stripe determines if a loader UI should be shown).

**currency** string

Used with the [Payment Element](#).

Influences available payment methods when creating [SetupIntents](#) with

available payment methods when creating SetupIntents with

**automatic\_payment\_methods** . Payment Element renders the payment methods enabled in the [Stripe Dashboard](#) that support the provided currency.

Three-letter ISO currency code, in lowercase. Must be a [supported currency](#).

## stripe.elements(options?)

This method creates an [Elements](#) instance, which manages a group of elements.

Use `stripe.elements()` with `mode`, `currency`, and `amount` when initializing the Payment Element without an Intent. Refer to [Collect payment details before creating an Intent](#) for further details.

### Method parameters

#### options optional object

A set of options to create this [Elements](#) instance with.

[× Hide options properties](#)**fonts** array

An array of custom fonts, which elements created from the `Elements` object can use. Fonts can be specified as [CssFontSource](#) or [CustomFontSource](#) objects.

**locale** string

A **locale** to display placeholders and error strings in. Default is `auto` (Stripe detects the locale of the browser).

Setting the locale does not affect the behavior of postal code validation—a valid postal code for the billing country of the card is still required.

**mode** CONDITIONALLY REQUIRED 'payment' | 'setup' | 'subscription'

Required to use with the [Payment Element](#).

Filters out payment methods based on intended use.

**currency** CONDITIONALLY REQUIRED string

Required to use with the [Payment Element](#).

Three-letter [ISO currency code](#), in lowercase. Must be a [supported currency](#).

**amount** CONDITIONALLY REQUIRED number

Used with the [Payment Element](#). Required when `mode` is `payment` or `subscription`.

Shown in Apple Pay, Google Pay, or Buy now pay later UIs. The amount intended to be collected from the customer right now. A positive integer representing how much to charge in the [smallest currency unit](#) (e.g., 100 cents to charge \$1.00 or 100 to charge ¥100, a zero-decimal currency). The minimum amount is \$0.50 US or [equivalent in charge currency](#). The amount value supports up to eight digits (e.g., a value of 99999999 for a USD charge of \$999,999.99).

**setupFutureUsage** 'on\_session' | 'off\_session'

Used with the [Payment Element](#).

Indicates that you intend to make future payments with the payment details collected by the Payment Element.

This should match the `setup_future_usage` provided on the Intent used when confirming payment.

**captureMethod** 'automatic' | 'automatic\_async' | 'manual'

Used with the [Payment Element](#).

Influences available payment methods. This should match the `capture_method` provided on the Intent used when confirming payment.

**onBehalfOf** CONNECT ONLY string

Used with the [Payment Element](#).

The Stripe account ID which is the business of record. See [use cases](#) to determine if this option is relevant for your integration. This should match the `on_behalf_of` provided on the Intent used when confirming payment.

**paymentMethodTypes** array

Used with the [Payment Element](#).

A list of payment method types to render. You can omit this attribute to manage your payment methods from the [Stripe Dashboard](#).

**paymentMethodConfiguration** string

Used with the [Payment Element](#).

The `payment method configuration` to use when managing your payment methods from the [Stripe Dashboard](#). If none is specified, your default configuration is used.

**paymentMethodCreation** 'manual'

Used with the [Payment Element](#).

Allows PaymentMethods to be created from the Elements instance using `stripe.createPaymentMethod`.

**paymentMethodOptions** object

Used with the [Payment Element](#).

Additional payment-method-specific options for configuring Payment Element behavior when initialized without an intent.

+ Show paymentMethodOptions properties

**appearance** object

Supported for the [Payment Element](#), the [Link Authentication Element](#), and the [Address Element](#).

Match the design of your site with the [appearance option](#). The layout of each Element stays consistent, but you can modify colors, fonts, borders, padding, and more.

**loader** 'auto' | 'always' | 'never'

Supported for the [Payment Element](#), the [Link Authentication Element](#), and the [Address Element](#).

Display skeleton loader UI while waiting for Elements to be fully loaded, after they are mounted. Default is 'auto' (Stripe determines if a loader UI should be shown).

## `elements.update(options)`

This method updates options on an existing instance of `Elements`. Note that not all options (e.g. `fonts`) are updatable.

### Method parameters

**options** REQUIRED object

A set of options to update this `Elements` instance with.

[× Hide options properties](#)**locale** string

A **locale** to display placeholders and error strings in. Default is `auto` (Stripe detects the locale of the browser).

Setting the locale does not affect the behavior of postal code validation—a valid postal code for the billing country of the card is still required.

**mode** 'payment' | 'setup' | 'subscription'

Used with the [Payment Element](#).

Filters out payment methods based on intended use.

**currency** string

Used with the [Payment Element](#).

Three-letter [ISO currency code](#), in lowercase. Must be a [supported currency](#).

**amount** number

Used with the [Payment Element](#).

Shown in Apple Pay, Google Pay, or Buy now pay later UIs. The amount intended to be collected from the customer right now. A positive integer representing how much to charge in the **smallest currency unit** (e.g., 100 cents to charge \$1.00 or 100 to charge ¥100, a zero-decimal currency). The minimum amount is \$0.50 US or [equivalent in charge currency](#). The amount value supports up to eight digits (e.g., a value of 99999999 for a USD charge of \$999,999.99).

**setupFutureUsage** 'on\_session' | 'off\_session'

Used with the [Payment Element](#).

Influences available payment methods and the **terms** shown by the Payment Element.

This should match the [setup\\_future\\_usage](#) provided on the Intent used when confirming payment.

**captureMethod** 'automatic' | 'automatic\_async' | 'manual'

Used with the [Payment Element](#).

Influences available payment methods. This should match the [capture method](#) provided on the PaymentIntent or SetupIntent used when confirming payment.

**onBehalfOf** CONNECT ONLY string

Used with the [Payment Element](#).

The Stripe account ID which is the business of record. See [use cases](#) to determine if this option is relevant for your integration. This should match the [on\\_behalf\\_of](#) provided on the Intent used when confirming payment.

**paymentMethodTypes** array

Used with the [Payment Element](#).

Instead of using dynamic payment methods, declare specific [payment methods](#) to enable.

**paymentMethodOptions** object

Used with the [Payment Element](#).

Additional payment-method-specific options for configuring Payment Element behavior when initialized without an intent.

+ Show paymentMethodOptions properties

**appearance** object

Used with the [Payment Element](#).

Match the design of your site with the [appearance option](#). The layout of each Element stays consistent, but you can modify colors, fonts, borders, padding, and more.

## element.submit()

Use `elements.submit()` when creating the [Elements object without an Intent](#).

Before confirming a payment, call `elements.submit()` to validate the form fields and collect any data required for [wallets](#).

### Returns

---

This method returns a [Promise](#) which resolves with a result object. If this method succeeds, the result object will be empty. If this method fails, the result object will contain a localized error message in the `error.message` field.

---

## The Element

Use `Element` instances to collect sensitive information in your checkout flow.

---

## The Payment Element

The [Payment Element](#) is an embeddable component for securely collecting payment details. The Payment Element supports dozens of payment methods with a single

integration.

---

```
elements.create('payment', options?)
```

This method creates an instance of the Payment Element.

## Method parameters

---

### **type** REQUIRED 'payment'

The type of Element being created, which is `payment` in this case.

---

### **options** optional object

Options for creating the Payment Element.

## [X Hide options properties](#)

### **layout** 'accordion' | 'tabs' | object

Specify the layout for the Payment Element. If you only pass a layout type ( 'accordion' or 'tabs' ) without any additional parameters, the Payment Element renders using that layout and the default values associated with it.

An object can also be passed to specify the layout with additional configuration.

[+ Show layout properties](#)

### **defaultValues** object

Provide initial customer information that will be displayed in the Payment Element. The form will render with empty fields if not provided.

[+ Show defaultValues properties](#)

### **business** object

Provide information about your business that will be displayed in the Payment Element. This information will be retrieved from your Stripe account if not provided.

[X Hide business properties](#)

#### **name** string

The name of your business. Your business name will be used to render mandate text for some payment methods.

### **paymentMethodOrder** array

By default, the Payment Element will use a dynamic ordering that optimizes payment method display for each user.

You can override the default order in which payment methods are displayed in the Payment Element with a list of payment method types.

If the associated PaymentIntent has payment method types not specified in

If the associated PaymentIntent has payment method types not specified in `paymentMethodOrder`, they will be displayed after the payment methods you specify. If you specify payment method types not on the associated PaymentIntent, they will be ignored.

#### **fields** object

By default, the Payment Element will collect all necessary details to complete a payment.

For some payment methods, this means that the Payment Element will collect details like name or email that you may have already collected from the user. If this is the case, you can prevent the Payment Element from collecting these data by using the `fields` option.

If you disable the collection of a certain field with the `fields` option, you must pass that same data to [stripe.confirmPayment](#) or the payment will be rejected.

See [below](#) for details.

+ Show fields properties

#### **readOnly** boolean

Applies a read-only state to the Payment Element so that payment details can't be changed. Default is false.

Enabling the `readOnly` option doesn't change the Payment Element's visual appearance. If you want to adjust the way the Payment Element looks, use the [Appearance API](#).

#### **terms** object

Control how mandates or other legal agreements are displayed in the Payment Element. Use `never` to never display legal agreements. The default setting is `auto`, which causes legal agreements to only be shown when necessary.

+ Show terms properties

#### **wallets** object

By default, the Payment Element will display all the payment methods that the underlying Payment Intent was created with.

However, wallets like Apple Pay and Google Pay are not payment methods per the Payment

Intent API. They will show when the Payment Intent has the `card` payment method and the customer is using a supported platform and have an active card in their account. This is the `auto` behavior, and it is the default for choice for all wallets.

If you do not want to show a given wallet as a payment option, you can set its property in `wallets` to `never`.

+ Show wallets properties

## Create the Payment Element with customized fields

### Option parameter

#### **fields** object

Pass an object to specify payment `fields` you don't want to collect with the Payment Element.

### `<div>` Hide fields properties

`billingDetails` 'never' | 'auto' | object

Specify `never` to avoid collecting all **billing details** in the Payment Element. If you would like to disable only certain billing details, pass an object specifying which fields you would like to disable collection for. The default setting for each field or object is `auto`.

#### `<div>` Hide `billingDetails` properties

`name` 'never' | 'auto'

`email` 'never' | 'auto'

`phone` 'never' | 'auto'

`address` 'never' | 'auto' | object

`+ Show address properties`

---

`elements.getElement('payment')`

This method retrieves a previously created Payment Element.

### Method parameters

`type` REQUIRED 'payment'

The type of Element being retrieved, which is `payment` in this case.

## Returns

---

`elements.getElement('payment')` returns one of the following:

- An instance of a Payment Element.
  - `null`, when no Payment Element has been created.
- 

## `element.update(options)`

Updates the options the **Payment Element** was initialized with. Updates are merged into the existing configuration.

**NOTE:** Don't use `element.update()` to fetch updates from a **PaymentIntent** or **SetupIntent**. Use `elements.fetchUpdates()` instead.

### Method parameters

---

#### `options` optional object

Options for updating the Payment Element.

[× Hide options properties](#)**defaultValues object**

Provide initial customer information that will be displayed in the Payment Element. The form will render with empty fields if not provided.

[+ Show defaultValues properties](#)**business object**

Provide information about your business that will be displayed in the Payment Element. This information will be retrieved from your Stripe account if not provided.

[× Hide business properties](#)**name string**

The name of your business. Your business name will be used to render mandate text for some payment methods.

**paymentMethodOrder array**

By default, the Payment Element will use a dynamic ordering that optimizes payment method display for each user.

You can override the default order in which payment methods are displayed in the Payment Element with a list of payment method types.

If the associated PaymentIntent has payment method types not specified in

`paymentMethodOrder`, they will be displayed after the payment methods you specify. If you specify payment method types not on the associated PaymentIntent, they will be ignored.

**fields object**

By default, the Payment Element will collect all necessary details to complete a payment.

For some payment methods, this means that the Payment Element will collect details like name or email that you may have already collected from the user. If this is the case, you can prevent the Payment Element from collecting these data by using the `fields` option

Prevent the Payment Element from collecting these data by using the `fields` option.

If you disable the collection of a certain field with the `fields` option, you must pass that same data to `stripe.confirmPayment` or the payment will be rejected.

See [below](#) for details.

+ Show fields properties

#### `readOnly` boolean

Applies a read-only state to the Payment Element so that payment details can't be changed. Default is false.

Enabling the `readOnly` option doesn't change the Payment Element's visual appearance. If you want to adjust the way the Payment Element looks, use the [Appearance API](#).

#### `terms` object

Control how mandates or other legal agreements are displayed in the Payment Element. Use `never` to never display legal agreements. The default setting is `auto`, which causes legal agreements to only be shown when necessary.

+ Show terms properties

## `elements.fetchUpdates()`

Used with the [Payment Element](#).

This method fetches updates from the associated [PaymentIntent](#) or [SetupIntent](#) on an existing instance of `Elements`, and reflects these updates in the Payment Element.

## Returns

---

This method returns a `Promise` which resolves with a result object. If this method succeeds, the result object will be empty. If this method fails, the result object will contain a localized error message in the `error.message` field. If the associated `PaymentIntent` or `SetupIntent` is in an unexpected status, the result object will also contain the intent's status in the `error.status` field.

---

### `element.collapse()`

This method collapses the Payment Element into a row of payment method tabs.

---

## The Express Checkout Element

The Express Checkout Element is an embeddable component for accepting payments through one-click payment buttons.

---

### `elements.create('expressCheckout', options?)`

This method creates an instance of the Express Checkout Element.

## Method parameters

---

**type** REQUIRED 'expressCheckout'

The type of Element being created, which is `expressCheckout` in this case.

---

**options** optional object

Options for creating the Express Checkout Element.

## [X Hide options properties](#)

### **buttonHeight** number

By default, the height of the buttons are 44px.

You can override this to specify a custom button height in the range of 40px-55px.

### **buttonTheme** object

Specify the preferred button theme to use. By default, Elements determines the themes based on the specified [appearance option](#).

[+ Show buttonTheme properties](#)

### **buttonType** object

Specify the preferred button type to display.

[+ Show buttonType properties](#)

### **layout** object

Specify how the buttons are arranged in a grid-like layout in the Express Checkout Element. Elements determines the layout by using certain factors, such as available space, number of buttons, and the defined `layout` object.

[+ Show layout properties](#)

### **paymentMethodOrder** array

By default, the Express Checkout Element uses a dynamic ordering that optimizes payment method display for each user.

You can override the default order in which payment methods display in the Express Checkout Element with a list of payment method types.

If there are payment methods that will show that are not specified in `paymentMethodOrder`, they display after the payment methods you specify. If you specify payment methods that will not show, they are ignored.

With this configuration, wallets like Apple Pay and Google Pay will be shown if supported by the customer's browser. If they're not supported or if the customer is using an unsupported browser, they'll be ignored.

### wallets object

By default, the Express Checkout Element displays all payment methods possible as a result of your Dashboard configuration.

However, wallets like Apple Pay and Google Pay have additional configurations that Stripe can show them. These wallets show when the customer is using a supported platform and when we determine it's advantageous for your conversion. This is the `auto` behavior, and it's the default choice for all wallets.

If you want to always show the wallets when the customer is using a supported platform, you can set its property in `wallets` to `always`.

If you don't want to show a given wallet as a payment option, set its property in `wallets` to `never`.

[+ Show wallets properties](#)

## `elements.getElement('expressCheckout')`

This method retrieves a previously created Express Checkout Element.

### Method parameters

**type** REQUIRED 'expressCheckout'

The type of Element being retrieved, which is `expressCheckout` in this case.

## Returns

---

`elements.getElement('expressCheckout')` returns one of the following:

- An instance of an Express Checkout Element.
  - `null`, when no Express Checkout Element has been created.
- 

## `element.update(options)`

Updates the options the **Express Checkout Element** was initialized with. Updates merge into the existing configuration.

### Method parameters

---

#### **options** optional object

Options for updating the Express Checkout Element.

[X Hide options properties](#)**layout object**

Specify how the buttons are arranged in a grid-like layout in the Express Checkout Element. Elements determines the layout by using certain factors, such as available space, number of buttons, and the defined `layout` object.

[+ Show layout properties](#)**paymentMethodOrder array**

By default, the Express Checkout Element uses a dynamic ordering that optimizes payment method display for each user.

You can override the default order in which payment methods display in the Express Checkout Element with a list of payment method types.

If there are payment methods that will show that are not specified in `paymentMethodOrder`, they display after the payment methods you specify. If you specify payment methods that will not show, they are ignored.

---

**expressCheckoutElement.on('click', handler)**

The `click` event is triggered from an Express Checkout Element when the customer clicks a payment button. Use this event to configure the payment interface.

## Method parameters

---

**event** REQUIRED 'click'

The name of the event. In this case, `click`.

#### **handler** REQUIRED function

`handler(event) => void` is a **callback function** you provide that's called after the event is fired.

After it's called, it passes an event object with the following properties:

##### `x Hide handler event object properties`

###### **elementType** REQUIRED 'expressCheckout'

The type of element the event is fired from, which is `expressCheckout` in this case.

###### **expressPaymentType** REQUIRED 'apple\_pay' | 'google\_pay' | 'paypal' | 'link'

The payment method the customer checks out with.

#### **resolve** REQUIRED function

A function `resolve(payload) => void` that's called to show the payment interface. You must call this function within 1 second if you handle the `click` event.

`+ Show resolve parameters`

## `expressCheckoutElement.on('confirm', handler)`

The `confirm` event is triggered from an Express Checkout Element when the customer finalizes their payment. Use this event to trigger payment confirmation.

### Method parameters

**event** REQUIRED string

The name of the event. In this case, `confirm`.

**handler** REQUIRED function

A callback function `handler(event) => void` you provide that's called after the event is fired.

When called, it passes an event object with the following properties:

### [× Hide handler event object properties](#)

#### **elementType** REQUIRED 'expressCheckout'

The type of element the event fires from, which is `expressCheckout` in this case.

#### **expressPaymentType** REQUIRED 'apple\_pay' | 'google\_pay' | 'paypal' | 'link'

The payment method the customer checks out with.

#### **paymentFailed** REQUIRED function

A function `paymentFailed(payload) => void` that's called if you're unable to process the customer's payment.

[+ Show paymentFailed parameters](#)

#### **billingDetails** object

Object containing information about the customer's billing details.

[+ Show billingDetails properties](#)

#### **shippingAddress** object

Object containing information about the customer's shipping address.

[+ Show shippingAddress properties](#)

#### **shippingRate** object

Object containing information about the selected shipping rate.

[+ Show shippingRate properties](#)

## expressCheckoutElement.on('cancel', handler)

The `cancel` event is triggered from an Express Checkout Element when the payment interface is dismissed.

Note that in some browsers, the payment interface might be dismissed by the customer even after they authorize the payment. This means that you might receive a `cancel` event after receiving a `confirm` event. If you're using the `cancel` event as a hook for canceling the customer's order, make sure you also refund the payment that you just created.

### Method parameters

---

#### event REQUIRED string

The name of the event. In this case, `cancel`.

---

#### handler REQUIRED function

A callback function that you provide that's called after the event is fired.

---

## expressCheckoutElement.on('shippingaddresschange', handler)

The `shippingaddresschange` event is triggered from an Express Checkout Element whenever the customer selects a new address in the payment interface.

### Method parameters

---

#### event REQUIRED string

The name of the event. In this case, `shippingaddresschange`.

#### **handler** REQUIRED function

A callback function `handler(event) => void` you provide that's called after the event is fired.

After it's called, it passes an event object with the following properties:

##### **x Hide handler event object properties**

#### **elementType** REQUIRED 'expressCheckout'

The type of element the event is fired from, which is `expressCheckout` in this case.

#### **resolve** REQUIRED function

A function `resolve(payload) => void` that's called if the recipient's shipping address is valid.

+ Show resolve parameters

#### **reject** REQUIRED function

A function `reject() => void` that's called if the recipient's shipping address is invalid.

#### **name** string

The name of the recipient.

#### **address** string

The shipping address of the recipient.

+ Show address properties

## expressCheckoutElement.on('shippingratechange', handler)

The `shippingratechange` event is triggered from an Express Checkout Element whenever the customer selects a new shipping rate in the payment interface.

### Method parameters

---

#### event REQUIRED string

The name of the event. In this case, `shippingratechange`.

---

#### handler REQUIRED function

A callback function `handler(event) => void` you provide that's called after the event is fired.

After it's called, it passes an event object with the following properties:

[× Hide handler event object properties](#)**elementType** REQUIRED 'expressCheckout'

The type of element the event is fired from, which is `expressCheckout` in this case.

**resolve** REQUIRED function

A function `resolve(payload) => void` that's called if the customer's shipping rate is valid.

[+ Show resolve parameters](#)**reject** REQUIRED function

A function `reject() => void` that's called if the customer's shipping rate is invalid.

**shippingRate** object

The shipping rate selected by the customer.

[+ Show shippingRate properties](#)

## The Link Authentication Element

The [Link Authentication Element](#) is an embeddable component for collecting email addresses and allow users to log into Link on your checkout page.

## elements.create('linkAuthentication', options?)

This method creates an instance of the Link Authentication Element.

### Method parameters

#### **type** REQUIRED 'linkAuthentication'

The type of Element being created, which is `linkAuthentication` in this case.

#### **options** optional object

Options for creating the Link Authentication Element.

[X Hide options properties](#)

##### **defaultValues** object

Provide the initial contact information that will be displayed in the Link Authentication Element. The form will render with empty fields if not provided.

[+ Show defaultValues properties](#)

## elements.getElement('linkAuthentication')

This method retrieves a previously created Link Authentication Element.

### Method parameters

**type REQUIRED** 'linkAuthentication'

The type of Element being retrieved, which is `linkAuthentication` in this case.

## Returns

---

`elements.getElement('linkAuthentication')` returns one of the following:

- An instance of a Link Authentication Element.
  - `null`, when no Link Authentication Element has been created.
- 

## The Address Element

The Address Element is an embeddable component for collecting local and international billing and shipping addresses.

---

`elements.create('address', options)`

This method creates an instance of the Address Element.

### Method parameters

---

**type REQUIRED** 'address'

The type of Element being created, which is `address` in this case.

---

#### **options REQUIRED object**

Options for creating the Address Element.

## [× Hide options properties](#)

**mode** REQUIRED 'shipping' | 'billing'

Specify which mode you would like to use Address Element for.

When `shipping` mode is used with the Payment Element and Link Authentication Element, it will automatically pass shipping information when confirming Payment Intent or Setup Intent.

When `billing` mode is used with the Payment Element, it will automatically pass the billing information when confirming Payment Intent or Setup Intent.

## **autocomplete** object

By default, the Address Element will have autocomplete enabled with Stripe provided Google Maps API key for certain countries if any of the following condition is met:

- If Payment Element is mounted in the same elements group as Address Element in a single page application.
- If the Address Element is used in an active Link session.

You can customize the autocomplete setting with this option.

## [× Hide autocomplete properties](#)

**mode** REQUIRED 'automatic' | 'disabled' | 'google\_maps\_api'

Specify `disabled` to disable autocomplete in the Address Element.

Specify `google_maps_api` to enable [Google Maps API](#) with your own key. It will only be used when Stripe provided Google Maps API key is not available.

The default setting is `automatic`, where we'll support autocomplete when possible.

## **apiKey** string

Specify your own [Google Maps API key](#) with it.

**Only needs to be passed in when `autocomplete.mode` is set to `google_maps_api`.**

**allowedCountries** array

By default, the Address Element will display all countries for selection.

You can specify which countries are displayed in the Address Element with a list of two-letter country codes. If only one country is specified, the country field will not display.

**blockPoBox** boolean

By default, PO boxes are considered a valid address type.

You can override this to invalidate PO Boxes.

**contacts** array

An array of **Contact** that can be displayed as saved addresses in the Address Element. The first contact will be automatically selected.

**defaultValues** object

Provide the initial information that will be displayed in the Address Element. The form will render with empty fields if not provided.

× Hide **defaultValues** properties**name** string

Provide the initial full name or organization name.

**firstName** string

Provide the initial first name. The **display.name** option must be set to `split` if this property is specified.

**lastName** string

Provide the initial last name. The **display.name** option must be set to `split` if this property is specified.

**phone** string

Provide the initial phone number value. The **fields.phone** option must be set to `always`

If this property is specified.

#### address object

Provide the initial address details.

+ Show address properties

#### fields object

By default, the Address Element will collect all the necessary information needed for an address. In some cases, it might be necessary to collect other types of information. You can specify other types of fields to render in the form with this option.

× Hide fields properties

##### phone 'always' | 'auto' | 'never'

Specify `always` to enable phone number collection in the Address Element. Only collect phone numbers if you need them for the transaction. Default is `auto`.

#### validation object

By default, the Address Element will enforce preset validation for each field. You can customize the settings by using this option.

× Hide validation properties

#### phone object

× Hide phone properties

##### required 'always' | 'auto' | 'never'

Specify `always` to make phone number a required field. The `fields.phone` option must be set to `always` if this property is specified. Default is `auto`.

#### display object

You can customize how certain fields are displayed.

#### **x Hide display properties**

`name 'full' | 'split' | 'organization'`

By default, the Address Element will display a full name field. Specify 'split' to display a first name field and a last name field. Specify 'organization' to display an organization field.

---

`elements.getElement('address')`

This method retrieves a previously created Address Element.

## Method parameters

---

**type** REQUIRED 'address'

The type of Element being retrieved, which is `address` in this case.

## Returns

---

`elements.getElement('address')` returns one of the following:

- An instance of an Address Element.
- `null`, when no Address Element has been created.

---

## `element.update(options)`

Updates the options the **Address Element** was initialized with. Updates are merged into the existing configuration.

## Method parameters

---

**options** optional object

Options for updating the Address Element.

## `<div>` × Hide options properties

### `fields` object

By default, the Address Element will collect all the necessary information needed for an address. In some cases, it might be necessary to collect other types of information. You can specify other types of fields to render in the form with this option.

#### `<div>` × Hide fields properties

##### `phone` 'always' | 'auto' | 'never'

Specify `always` to enable phone number collection in the Address Element. Only collect phone numbers if you need them for the transaction. Default is `auto`.

### `validation` object

By default, the Address Element will enforce preset validation for each field. You can customize the settings by using this option.

#### `<div>` × Hide validation properties

##### `phone` object

###### `<div>` × Hide phone properties

###### `required` 'always' | 'auto' | 'never'

Specify `always` to make phone number a required field. The `fields.phone` option must be set to `always` if this property is specified. Default is `auto`.

## `element.getValue()`

Validates and retrieves form values from an Address Element. If there are any input validation errors, the errors will display by their respective fields.

## Returns

---

`addressElement.getValue()` returns a promise. This promise will return an object with the following:

- `complete`, `true` if the value is well-formed and potentially complete. The `complete` value can be used to progressively disclose the next parts of your form or to enable form submission. It is not an indicator of whether a customer is done with their input—it only indicates that the Element contains a potentially complete, well-formed value. In many cases the customer could still add further input.
- `isNewAddress`, `true` if the Address Element is currently displaying the form collection view.
- `value`, an object containing the current address information. The `firstName` and `lastName` properties only appear if the `display.name` option is set to `split`. The `phone` property only appears if the `fields.phone` option is set to `always`.

---

## Issuing Elements

[Issuing Elements](#) allows you to display the sensitive data of your Issuing cards in a PCI-compliant manner.

```
elements.create('issuingCardNumberDisplay', options)
```

This method creates an instance of an individual Issuing Element.

It takes the `type` of Element to create as well as an `options` object.

## Method parameters

### `type` REQUIRED string

The type of Element being created. One of: `issuingCardNumberDisplay`, `issuingCardCvcDisplay`, `issuingCardExpiryDisplay`, `issuingCardPinDisplay`, or `issuingCardCopyButton`.

### `options` REQUIRED object

Options for creating an Issuing element.

#### `x Hide options properties`

##### `issuingCard` string

ID of the `Issuing card` to be displayed in this Element.

##### `ephemeralKeySecret` string

The `secret` component of the ephemeral key created to display this Element.

##### `nonce` string

The ephemeral key nonce used to create the ephemeral key provided to this Element.

##### `style` object

Customize the appearance of this Element using CSS properties passed in a `Style` object.

## Other Elements

Stripe also offers a [set of Elements for individual payment methods](#) that you can use in your payment flows.

---

```
elements.create('card', options?)
```

This method creates an instance of an individual `Element`.

It takes the `type` of `Element` to create as well as an `options` object.

### Method parameters

---

`type` **REQUIRED** `'card'`

The type of element you are creating. In this case, `card`.

---

`options` **optional** `object`

Options for creating a `card` element.

## [X Hide options properties](#)

### **classes** object

Set custom class names on the container DOM element when the Stripe element is in a particular state.

[+ Show classes properties](#)

### **style** object

Customize the appearance of this element using CSS properties passed in a [Style](#) object.

### **value** string

A pre-filled set of values to include in the input (e.g., `{postalCode: '94110'}` ). Note that sensitive card information (card number, CVC, and expiration date) cannot be pre-filled

### **hidePostalCode** boolean

Hide the postal code field. Default is `false`. If you are already collecting a full billing address or postal code elsewhere, set this to `true`.

### **iconStyle** string

Appearance of the icon in the Element. Either `solid` or `default`.

### **hideIcon** boolean

Hides the icon in the Element. Default is `false`.

### **disabled** boolean

Applies a disabled state to the Element such that user input is not accepted. Default is `false`.

### **disableLink** boolean

Disables and hides the Link button in the Element. Default is `false`. You can also disable Link across all instances of `card` and `cardNumber` elements in your [payment method](#)

## settings.

### elements.getElement(type)

This method looks up a previously created **Element** by its type.

#### Method parameters

**type** REQUIRED string

The type of **Element** to lookup.

#### Returns

`elements.getElement` returns one of the following:

- An instance of an **Element** with a matching type.
- `null`, when no **Element** with a matching type has been created.

### card .update(options)

Updates the options the **Element** was initialized with. Updates are merged into the existing configuration.

If you collect certain information in a different part of your interface (e.g., ZIP or postal code), use `element.update` with the appropriate information.

The styles of an `Element` can be dynamically changed using `element.update`. This method can be used to simulate CSS media queries that automatically adjust the size of elements when viewed on different devices.

## Method parameters

---

### **options** optional object

Options for updating a `card` element.

## [X Hide options properties](#)

### **classes** object

Set custom class names on the container DOM element when the Stripe element is in a particular state.

[+ Show classes properties](#)

### **style** object

Customize the appearance of this element using CSS properties passed in a [Style](#) object.

### **value** string

A pre-filled set of values to include in the input (e.g., `{postalCode: '94110'}` ). Note that sensitive card information (card number, CVC, and expiration date) cannot be pre-filled

### **hidePostalCode** boolean

Hide the postal code field. Default is `false`. If you are already collecting a full billing address or postal code elsewhere, set this to `true`.

### **iconStyle** string

Appearance of the icon in the Element. Either `solid` or `default`.

### **hideIcon** boolean

Hides the icon in the Element. Default is `false`.

### **disabled** boolean

Applies a disabled state to the Element such that user input is not accepted. Default is `false`.

## Style the Element container

Style the container you mount an `Element` to as if it were an `<input>` on your page.

For example, to control `padding` and `border` on an `Element`, set these properties on the container. This is usually done by re-using the classes that you have applied to your DOM `<input>` elements. After the `Element` is mounted, the

`.StripeElement` class is added to the container. Additionally, the following classes are automatically added to the container when the `Element` is complete, empty, focused, invalid, or autofilled by the browser:

- `.StripeElement--complete`
- `.StripeElement--empty`
- `.StripeElement--focus`
- `.StripeElement--invalid`
- `.StripeElement--webkit-autofill` (Chrome and Safari only)

These class names can be customized using the `classes` option when you [create an Element](#).

---

### `element.mount(domElement)`

The `element.mount` method attaches your `Element` to the DOM. `element.mount` accepts either a CSS Selector (e.g., `'#card-element'`) or a DOM element.

You need to create a container DOM element to mount an `Element`. If the container DOM element has a label, the `Element` is automatically focused when its label is clicked. There are two ways to do this:

- Mount the instance within a `<label>`.
- Create a `<label>` with a `for` attribute, referencing the ID of your container.

## Method parameters

---

`domElement` **REQUIRED** string | DOM element

The CSS selector or DOM element where your `Element` will be mounted.

---

## Element methods

Below are a number of methods that are in common between all `Element` UIs.

---

`element.blur()`

Blurs the `Element`.

---

`element.clear()`

Clears the value(s) of the [Element](#).

---

### `element.destroy()`

Removes the [Element](#) from the DOM and destroys it. A destroyed [Element](#) can not be re-activated or re-mounted to the DOM.

---

### `element.focus()`

Focuses the [Element](#).

 This method will currently not work on iOS 13+ due to a system limitation.

---

### `element.unmount()`

Unmounts the [Element](#) from the DOM. Call `element.mount` to re-attach it to the DOM.

---

## Element events

The only way to communicate with your **Element** is by listening to an event. An Element might emit any of the events below. All events have a payload object that has an `elementType` property with the type of the **Element** that emitted the event.

---

**paymentElement**  `.on('change', handler)`

The change event is triggered when the **Element**'s value changes. The event payload always contains certain keys, in addition to some **Element**-specific keys.

## Method parameters

---

**event** **REQUIRED** `'change'`

The name of the event. In this case, `change`.

---

**handler** **REQUIRED** `function`

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired. When called it will be passed an event object with the following properties:

[x Hide handler event object properties](#)

**elementType** string

The type of element that emitted this event.

**empty** boolean

`true` if the value is empty.

**complete** boolean

`true` if all required fields for the selected payment method in the Payment Element have been filled with potentially valid input.

**collapsed** boolean

`true` if the Payment Element is currently collapsed

**value** object

An object containing the current selected PaymentMethod type.

-  Consult with your legal counsel regarding your requirements and obligations about how you collect, use, and store customers' personal data

## `element.on('ready', handler)`

Triggered when the `Element` is fully rendered and can accept `element.focus` calls.

### Method parameters

**event** REQUIRED 'ready'

The name of the event. In this case, `ready`.

**handler** REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired.

After it's called, it passes an event object with the following properties:

[× Hide handler event object properties](#)

**elementType** REQUIRED string

The type of element the event is fired from.

**availablePaymentMethods** object | undefined

This field is **only** present on the `expressCheckout` Element. Describes which buttons render in the Element. Returns undefined if no buttons will render.

[+ Show availablePaymentMethods properties](#)

---

`element.on('focus', handler)`

Triggered when the `Element` gains focus.

## Method parameters

---

**event** REQUIRED 'focus'

The name of the event. In this case, `focus`.

---

#### **handler** REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired.

---

## `element.on('blur', handler)`

Triggered when the `Element` loses focus.

### Method parameters

---

#### **event** REQUIRED 'blur'

The name of the event. In this case, `blur`.

---

#### **handler** REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired.

---

## `element.on('escape', handler)`

Triggered when the escape key is pressed within an Element.

## Method parameters

### event REQUIRED 'escape'

The name of the event. In this case, `escape`.

### handler REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired.

## `expressCheckoutElement` .`on('click', handler)`

The `click` event is triggered from an Express Checkout Element when the customer clicks a payment button. Use this event to configure the payment interface.

## Method parameters

### event REQUIRED 'click'

The name of the event. In this case, `click`.

### handler REQUIRED function

`handler(event) => void` is a **callback function** you provide that's called after the event is fired.

After it's called, it passes an event object with the following properties:

### × Hide handler event object properties

**elementType** REQUIRED 'expressCheckout'

The type of element the event is fired from, which is `expressCheckout` in this case.

**expressPaymentType** REQUIRED 'apple\_pay' | 'google\_pay' | 'amazon\_pay' | 'paypal' | 'link'

The payment method the customer checks out with.

**resolve** REQUIRED function

A function `resolve(payload) => void` that's called to show the payment interface. You must call this function within 1 second if you handle the `click` event.

+ Show resolve parameters

---

`element.on('loaderror', handler)`

Triggered when the `Element` fails to load.

**This event is only emitted from the `payment`, `linkAuthentication`, `address`, and `expressCheckout` Elements.**

### Method parameters

**event** REQUIRED 'loaderror'

The name of the event. In this case, `loaderror`.

**handler** REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired.

When called it will be passed an event object with the following properties:

[X Hide handler event object properties](#)

**elementType** string

The type of element that emitted this event.

**error** object

An `error` object that describes the failure.

---

## `element.on('loaderstart', handler)`

Triggered when the **loader** UI is mounted to the DOM and ready to be displayed.

This event is only emitted from the `payment`, `linkAuthentication`, and `address` Elements.

### Method parameters

---

**event** REQUIRED 'loaderstart'

The name of the event. In this case, `loaderstart`.

---

**handler** REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired.

When called it will be passed an event object with the following properties:

[X Hide handler event object properties](#)

**elementType** string

The type of element that emitted this event.

**cardElement**  `.on('networkschange', handler)`

Triggered when there is a change to the available networks the provided card can run on. If the list of available networks is still loading, an event with `networks: null` and `loading: true` is triggered. When the list of available networks loads, Stripe triggers an additional event that contains the list of these networks and shows `loading: false`. Refer to our [card brand choice guide](#) for further details.

## Method parameters

**event** REQUIRED 'networkschange'

The name of the event. In this case, `networkschange`.

**handler** REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired. When called it will be passed an event object with the following properties:

### [× Hide handler event object properties](#)

#### **loading** boolean

`true` if the networks are loading. `false` when Stripe returns all the available networks.

#### **networks** stringArray | null

All available networks for the card number provided. `null` if the networks are still loading.

## Input validation

Stripe elements validate customer input as it is typed. To help your customers catch mistakes, listen to `change` events on an `Element` and display any errors.

## Postal code formatting

The `card` element automatically determines your customer's billing address country based on their card number. Using this information, the postal code field validation reflects whether that country uses numeric or alphanumeric-formatted postal codes, or if the country uses postal codes at all. For instance, if a U.S. card is entered, the postal code field only accepts a five-digit numeric value. If it's a UK card, an alphanumeric value can be provided instead.

Many of our test cards have a U.S. billing address country. When using these to test your payment form, you must also use a five-digit U.S. ZIP code (e.g., 12345). To test elements with other postal code formats, use our [international test card numbers](#).

## Checkout

**Checkout** is a low-code payment integration that creates a customizable payment form so you can quickly collect payments on desktop and mobile devices.

### `stripe.initEmbeddedCheckout(options)`

This method initializes embedded Checkout.

#### Method parameters

##### **options** REQUIRED object

Embedded Checkout initialization options.

[X Hide options properties](#)

**clientSecret** REQUIRED string

The **client secret** for the [Checkout Session](#).

**onComplete** function

An optional callback function `onComplete() => void` that is called on completion for Checkout Sessions with `redirect_on_completion: if_required`.

## Returns

This method returns a `Promise` which resolves with an embedded Checkout instance.

---

**checkout.mount(domElement)**

The `checkout.mount` method attaches Checkout to the DOM. `checkout.mount` accepts either a CSS Selector (e.g., `'#checkout'`) or a DOM element.

You need to create a container DOM element to mount Checkout.

## Method parameters

---

### `domElement` REQUIRED string | DOM element

The CSS selector or DOM element where Checkout will be mounted.

---

### `checkout.unmount()`

Unmounts Checkout from the DOM. Call `checkout.mount` to reattach it to the DOM.

---

### `checkout.destroy()`

Removes Checkout from the DOM and destroys it. Once destroyed, an embedded Checkout instance cannot be reattached to the DOM.

Call `checkout.initEmbeddedCheckout` to create a new embedded Checkout instance after unmounting the previous instance from the DOM.

# Payment Intents

Accept global payments online with the [Payment Intents APIs](#). For step-by-step instructions on using the Payment Intents APIs, see the [accept a payment guide](#).

The following Stripe.js methods are available to use as part of your integration.

---

## stripe.confirmPayment(options)

Use `stripe.confirmPayment` to confirm a [PaymentIntent](#) using data collected by the [Payment Element](#), or with manually provided data via `confirmParams`. When called, `stripe.confirmPayment` will attempt to complete any [required actions](#), such as authenticating your user by displaying a 3DS dialog or redirecting them to a bank authorization page. Your user will be redirected to the `return_url` you pass once the confirmation is complete.

---

### Method parameters

**options** REQUIRED object

[X Hide options properties](#)**elements** CONDITIONALLY REQUIRED object

The [Elements](#) instance used to create the Payment Element.

Required if you [collect payment details before creating an Intent](#). It's always required if you don't provide a `clientSecret`.

**clientSecret** CONDITIONALLY REQUIRED string

The PaymentIntent's client secret.

Required if you [collect payment details before creating an Intent](#). It's always required if you don't provide an `elements` instance containing a [client secret](#).

**confirmParams** object

Parameters that will be passed on to the Stripe API. Refer to the [Payment Intents API](#) for a full list of parameters.

[X Hide confirmParams properties](#)**return\_url** REQUIRED string

The url your customer is redirected to after they complete the payment.

You can use the following query parameters, `payment_intent` (the PaymentIntent's ID) or `payment_intent_client_secret` (the PaymentIntent's client secret), to retrieve the PaymentIntent's `status`. You can also append your own query parameters to the `return_url`, which persist through the redirect process.

**shipping** RECOMMENDED object

The [shipping details](#) for the payment, if collected.

**Note:** When the [Address Element](#) in shipping mode is being used, shipping address details are collected from the Address Element and passed to the PaymentIntents [confirm endpoint](#) as the `shipping` parameter. You can also include additional `shipping` fields, which will be merged with the data collected from the Element. Values passed here will override details collected by Elements.

Here will override details collected by elements.

#### **payment\_method** string

If collected previously, the ID of the payment method to attach to this PaymentIntent. This is mutually exclusive with the `elements` parameter.

#### **payment\_method\_data** object

When you call `stripe.confirmPayment`, payment details are collected from the Element and passed to the PaymentIntents [confirm endpoint](#) as the `payment_method_data` parameter. You can also include additional `payment_method_data` fields, which will be merged with the data collected from the Element.

+ Show payment\_method\_data properties

#### **expand** array

An array of pass through [PaymentIntent](#) expansion parameters ([learn more](#) ).

#### **redirect** 'always' | 'if\_required'

By default, `stripe.confirmPayment` will always redirect to your `return_url` after a successful confirmation. If you set `redirect: "if_required"`, then `stripe.confirmPayment` will only redirect if your user chooses a redirect-based payment method.

**Note:** Setting `if_required` requires that you handle successful confirmations for redirect-based and non-redirect based payment methods separately. When a non-redirect based payment method is successfully confirmed, `stripe.confirmPayment` will resolve with a `{paymentIntent}` object.

## Returns

`stripe.confirmPayment` will return a `Promise`. Upon a successful confirmation, your user will be redirected to the `return_url` you provide before the `Promise` ever resolves.

If the confirmation fails, the `Promise` will resolve with an `{error}` object that describes the failure. When the `error type` is `card_error` or `validation_error`, you can display the error message in `error.message` directly to your user. An error type of `invalid_request_error` could be due to an invalid request or 3DS authentication failures.

Note that for some payment methods such as iDEAL or Afterpay Clearpay, your user will first be redirected to an intermediate page to authorize the payment. If they fail to authorize the payment, they will be redirected back to your `return_url` and the `PaymentIntent` will have a `status` of `requires_payment_method`. In this case you should attempt to recollect payment from the user.

- i Note that `stripe.confirmPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a `PaymentIntent` by payment method

Below are a number of methods used to confirm a `PaymentIntent` for a specific payment method type.

## stripe.confirmCardPayment(`clientSecret`, `data?`, `options?`)

Use `stripe.confirmCardPayment` when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide and carry out 3DS or other next actions if they are required.

If you are using `Dynamic 3D Secure`, `stripe.confirmCardPayment` will trigger your Radar rules to execute and may open a dialog for your customer to authenticate their payment.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. It can also be called with an existing `PaymentMethod`, or if you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

#### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

#### `data` optional object

Data to be sent with the request. Refer to the `Payment Intents API` for a full list of parameters.

[× Hide data properties](#)

#### **payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

#### **shipping** RECOMMENDED object

The [shipping details](#) for the payment, if collected.

#### **return\_url** string

If you are [handling next actions yourself](#), pass in a `return_url`. If the subsequent action is `redirect_to_url`, this URL will be used on the return path for the redirect.

#### **receipt\_email** string

Email address that the receipt for the resulting payment will be sent to.

#### **setup\_future\_usage** string

Indicates that you intend to make future payments with this PaymentIntent's payment method.

If present, the payment method used with this PaymentIntent can be [attached to a Customer](#), even after the transaction completes.

Use `on_session` if you intend to only reuse the payment method when your customer is present in your checkout flow. Use `off_session` if your customer may or may not be in your checkout flow. See [saving card details during payment](#) to learn more.

Stripe uses `setup_future_usage` to dynamically optimize your payment flow and comply with regional legislation and network rules. For example, if your customer is impacted by [SCA](#), using `off_session` will ensure that they are authenticated while processing this PaymentIntent. You will then be able to collect [off-session payments](#) for this customer.

#### **payment\_method\_options** object

An object containing payment-method-specific configuration to confirm the

[PaymentIntent](#) with.

+ Show payment\_method\_options properties

#### options optional object

An options object to control the behavior of this method.

[× Hide options properties](#)

##### handleActions boolean

Set this to `false` if you want to [handle next actions yourself](#), or if you want to defer next action handling until later (e.g. for use in the [PaymentRequest API](#)). Default is `true`.

## Returns

`stripe.confirmCardPayment` will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent` : the successful [PaymentIntent](#) .
- `result.error` : an error. When the `error type` is `card_error` or `validation_error` , you can display the error message in `error.message` directly to your user. Refer to the [API reference](#) for other possible errors.

 Note that `stripe.confirmCardPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.confirmCardPayment` may trigger a **3D Secure** authentication challenge. This will be shown in a modal dialog and may be confusing for customers using assistive technologies like screen readers. You should make your form accessible by ensuring that success or error messages are clearly read out after this method completes.

## Confirm a card payment with payment data from an Element

Use `stripe.confirmCardPayment` with payment data from an **Element** by passing a `card` or `cardNumber` **Element** as `payment_method[card]` in the data argument.

The new `PaymentMethod` will be created with data collected by the `Element` and will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected by a `card` or `cardNumber` **Element**.

##### × Hide `payment_method` properties

###### `card` REQUIRED Element

Uses the provided `card` or `cardNumber` **Element** for confirmation.

###### `billing_details` RECOMMENDED object

The **billing\_details** associated with the card.

## Confirm a card payment with an existing payment method

Use `stripe.confirmCardPayment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

## Data argument properties

---

**payment\_method** REQUIRED string

The `id` of an existing **PaymentMethod**.

---

## Confirm a card payment with an existing token

For backwards compatibility, you can convert an existing `Token` into a `PaymentMethod` with `stripe.confirmCardPayment` by passing the `Token` to `payment_method[card][token]`. The newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

## Data argument properties

---

**payment\_method** REQUIRED object

Pass an object to confirm using an existing token.

× Hide payment\_method properties

**card** REQUIRED object

An object of card data.

× Hide card properties

**token** REQUIRED string

Converts the provided token into a `PaymentMethod` to use for confirmation.

**billing\_details** RECOMMENDED object

The `billing_details` associated with the card.

## Confirm a card payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmCardPayment` without passing in any additional data.

```
stripe.confirmUsBankAccountPayment(clientSecret, data?)
```

Use `stripe.confirmUsBankAccountPayment` in the **Accept a payment flow** for the **ACH Direct Debit** payment method to record the customer's authorization for payment.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. We suggest using `stripe.collectBankAccountForPayment`, which automatically collects bank account details and attaches a `PaymentMethod`. You may also choose to reuse an existing `PaymentMethod` or manually collect bank account details using the `data` parameter. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

#### `payment_method` object | string

The `id` of an existing `PaymentMethod` or an object of collected data. See use cases below for details.

## Returns

`stripe.confirmUsBankAccountPayment` will return a `Promise` which resolves with a result object. This object has either:

- `result.paymentIntent`: the successfully confirmed `PaymentIntent`.
- `result.error`: an error. Refer to the [API reference](#) and our [integration guide](#) for all possible errors.

## Confirm an ACH Direct Debit payment with an existing PaymentMethod

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmUsBankAccountPayment`.

### Data argument properties

---

`payment_method` REQUIRED string

The `id` of an existing `PaymentMethod` .

---

## Confirm an ACH Direct Debit payment with an attached PaymentMethod

If you have successfully called `stripe.collectBankAccountForPayment` or attached a `PaymentMethod` to this `PaymentIntent` already, then you can confirm the `PaymentIntent` without passing in any additional data.

---

## Confirm an ACH Direct Debit payment with self collected bank account information

If you already know the customer's bank account information, or want to collect it yourself, you can pass them in directly to create a new `PaymentMethod` and confirm the `PaymentIntent`.

### Data argument properties

---

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected.

**x Hide payment\_method properties**

**billing\_details REQUIRED Object**

The customer's **billing\_details** . `name` is required. Providing `email` allows your customer to receive [ACH Direct Debit mandate and microdeposit emails](#).

**x Hide billing\_details properties**

**name REQUIRED string**

The customer's name. The first and last name must be at minimum 2 characters each.

**email string**

The customer's email.

**us\_bank\_account REQUIRED Object**

The customer's [bank account information](#) .

**x Hide us\_bank\_account properties**

**account\_number REQUIRED string**

The customer's bank account number.

**routing\_number REQUIRED string**

The routing number of the customer's bank.

**account\_holder\_type REQUIRED string**

Account holder type: individual or company.

**account\_type string**

Account type: checkings or savings. Defaults to checking if omitted.

## stripe.confirmAcssDebitPayment(`clientSecret`, `data?`, `options?`)

Use `stripe.confirmAcssDebitPayment` in the [Accept a payment](#) flow for the [Canadian pre-authorized debit](#) payment method when the customer submits your payment form. When called, it will automatically load an on-page modal UI to collect bank account details and verification, accept a hosted mandate agreement, and confirm the [PaymentIntent](#) when the user submits the form. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a [PaymentIntent](#), it needs to have an attached [PaymentMethod](#).

`stripe.confirmAcssDebitPayment` automatically creates a new [PaymentMethod](#) for you when your customer completes the modal UI. It can also be called with an existing [PaymentMethod](#), which will load the modal UI to collect a new mandate agreement. These use cases are detailed in the sections that follow.

### Method parameters

#### `clientSecret` REQUIRED string

The [client secret](#) of the [PaymentIntent](#).

#### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

**payment\_method** object | string

The `id` of an existing PaymentMethod or an object of collected data. See use cases below for details.

**options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)

**skipMandate** boolean

Set this to `true` if you want to skip displaying the mandate confirmation.

## Returns

`stripe.confirmAcssDebitPayment` will return a `Promise` which resolves with a result object.

This object has either:

- `result.paymentIntent`: the successfully confirmed [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) and our [integration guide](#) for all possible errors.

**i** Note that `stripe.confirmAcssDebitPayment` may take several seconds to complete.

During that time, disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, show it to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Canadian pre-authorized debit payment with a new PaymentMethod

You can pass in the customer's billing details to create a new `PaymentMethod` and confirm the `PaymentIntent`. You are required to collect and include the customer's name and email address. This method loads an on-page modal UI that handles bank account details collection and verification, presents a hosted mandate agreement and collects authorization for you.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected.

##### `× Hide payment_method properties`

##### `billing_details` REQUIRED Object

The customer's `billing_details` . `name` and `email` are required.

##### `× Hide billing_details properties`

##### `name` REQUIRED string

The customer's name. The first and last name must be at minimum 2 characters each.

##### `email` REQUIRED string

The customer's email.

## Confirm a Canadian pre-authorized debit payment with an existing PaymentMethod

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmAcssDebitPayment`. This method loads an on-page modal UI that only presents a hosted mandate agreement and collects authorization for you.

### Data argument properties

---

`payment_method` REQUIRED string

The `id` of an existing `PaymentMethod` .

---

## Confirm a Canadian pre-authorized debit payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` without passing in any additional data. This method loads an on-page modal UI that only presents a hosted mandate agreement and collects authorization for you.

---

## Confirm a Canadian pre-authorized debit payment with self collected bank account information

If you already know the customer's bank account information, or want to collect it yourself, you can pass them in directly to create a new `PaymentMethod` and confirm the `PaymentIntent`. In this case, this method does not load the on-page modal UI, so you will need to [build your own mandate agreement page](#).

### Data argument properties

---

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected.

× Hide payment\_method properties

**billing\_details** REQUIRED Object

The customer's **billing\_details** . `name` and `email` are required.

× Hide billing\_details properties

**name** REQUIRED string

The customer's name. The first and last name must be at minimum 2 characters each.

**email** REQUIRED string

The customer's email.

**acss\_debit** REQUIRED Object

The customer's **bank account information** .

× Hide acss\_debit properties

**account\_number** REQUIRED string

The customer's bank account number.

**institution\_number** REQUIRED string

The institution number of the customer's bank.

**transit\_number** REQUIRED string

The transit number of the customer's bank.

## Confirm a Canadian pre-authorized debit payment with an existing PaymentMethod but skip mandate display

If you have already created a `PaymentMethod` and built your own mandate agreement page, you can reuse it by passing its `id` to `payment_method` when calling `stripe.confirmAcssDebitPayment` and skip the on-page modal UI at the same time.

### Data and options argument parameters

#### `data` object

Data to be sent with the request.

[X Hide data properties](#)

`payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

#### `options` object

An options object to control the behavior of this method.

[X Hide options properties](#)

`skipMandate` REQUIRED boolean

Set to `true` to skip the on-page modal UI.

## stripe.confirmAffirmPayment(`clientSecret`, `data?`, `options?`)

Use `stripe.confirmAffirmPayment` in the Affirm payment method creation flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`. When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

#### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

#### `data` optional object

Data to be sent with the request. Refer to the `Payment Intents API` for a full list of parameters.

[× Hide data properties](#)**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

**shipping** REQUIRED object

The [shipping details](#) for the payment.

[+ Show shipping properties](#)**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

**options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

By default, `stripe.confirmAffirmPayment` will trigger a redirect when successful. If there is an error, or when handling `next_action`s manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent` : the successful [PaymentIntent](#) .
- `result.error` : an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmAffirmPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm an Affirm payment without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass payment method billing details.

##### **x** Hide `payment_method` properties

#### `billing_details` RECOMMENDED object

The `billing_details` associated with the payment.

##### **+** Show `billing_details` properties

**shipping** REQUIRED object

The **shipping details** for the payment.

+ Show shipping properties

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

## Confirm an Affirm payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

#### `shipping` REQUIRED object

The `shipping details` for the payment.

+ Show shipping properties

#### `return_url` REQUIRED string

The url your customer will be directed to after they complete authentication.

## Confirm an Affirm payment with an attached PaymentMethod

If you have already attached a `return_url` and a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

```
stripe.confirmAfterpayClearpayPayment(  
  clientSecret,  
  data?,  
  options?  
)
```

Use `stripe.confirmAfterpayClearpayPayment` in the Afterpay Clearpay payment method creation flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`. When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

---

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[X Hide data properties](#)

**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

**options** optional object

An options object to control the behavior of this method.

[X Hide options properties](#)

**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

By default, `stripe.confirmAfterpayClearpayPayment` will trigger a redirect when successful. If there is an error, or when handling `next_action`s manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmAfterpayClearpayPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm an Afterpay Clearpay payment without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass payment method billing details.

**x Hide payment\_method properties**

#### `billing_details` REQUIRED object

The `billing_details` associated with the payment.

**+ Show billing\_details properties**

#### `shipping` RECOMMENDED object

The `shipping details` for the payment.

+ Show shipping properties

---

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

---

## Confirm an Afterpay Clearpay payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

---

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod`.

---

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

---

## Confirm an Afterpay Clearpay payment with an attached PaymentMethod

If you have already attached a `return_url` and a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

---

```
stripe.confirmAlipayPayment(clientSecret, data?, options?)
```

Use `stripe.confirmAlipayPayment` in the Alipay payment method creation flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`. When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

#### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

#### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

#### [X Hide data properties](#)

##### **payment\_method** RECOMMENDED string | object

The `id` of an existing [PaymentMethod](#). See the use case sections below for details.

##### **return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

#### **options** optional object

An options object to control the behavior of this method.

#### [X Hide options properties](#)

##### **handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

By default, `stripe.confirmAlipayPayment` will trigger a redirect when successful. If there is an error, or when handling `next_action`s manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.



Note that `stripe.confirmAlipayPayment` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a

waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm an Alipay payment Without an existing payment method

If you have not already created a `PaymentMethod`, the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm an Alipay payment With an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

---

## Confirm an Alipay payment With an attached PaymentMethod

If you have already attached a `return_url` and a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

---

### `stripe.confirmAuBecsDebitPayment(clientSecret, data?)`

Use `stripe.confirmAuBecsDebitPayment` in the **BECS Direct Debit Payments** with Payment Methods flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

---

### Method parameters

**clientSecret** REQUIRED string

The **client secret** of the `PaymentIntent`.

**data** optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[X Hide data properties](#)

**payment\_method** RECOMMENDED object | string

The `id` of an existing PaymentMethod or an object of collected data. See use cases below for details.

**setup\_future\_usage** string

To set up the BECS Direct Debit account for reuse, set this parameter to `off_session`. BECS Direct Debit only accepts an `off_session` value for this parameter. If a `customer` is provided on this `PaymentIntent`, the `PaymentMethod` will be attached to the customer when the `PaymentIntent` transitions to `processing`.

## Returns

`stripe.confirmAuBecsDebitPayment` will return a `Promise` which resolves with a result object.

This object has either:

- `result.paymentIntent`: the successful `PaymentIntent`.
- `result.error`: an error. Refer to the API reference for all possible errors.



Note that `stripe.confirmAuBecsDebitPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a BECS Debit payment with payment data from an Element

Create and attach a new PaymentMethod with `stripe.confirmAuBecsDebitPayment` by passing an `auBankAccount` [Element](#) to `payment_method[au_becs_debit]`. The new `PaymentMethod` will be created with the data collected by the [Element](#) and will be used to confirm the `PaymentIntent`. Additionally, to create a BECS Direct Debit `PaymentMethod`, you are required to collect and include the account holder's name and the customer's email address.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an `object` to confirm the payment using data collected by an `auBankAccount` [Element](#).

### **x Hide payment\_method properties**

#### **au\_becs\_debit REQUIRED Element**

An `auBankAccount` Element.

#### **billing\_details REQUIRED Object**

The customer's `billing_details` . `name` and `email` are required.

##### **x Hide billing\_details properties**

###### **name REQUIRED string**

The account holder's name.

###### **email REQUIRED string**

The customer's email.

## Confirm a BECS Debit payment with self collected data

If you already know the customer's BSB number and bank account number or want to collect it yourself, then you do not need to use the `auBankAccount` Element with

`stripe.confirmAuBecsDebitPayment` . You can pass in the customer's bank account information directly to create a new `PaymentMethod` and confirm the `PaymentIntent` . To create a BECS Direct Debit `PaymentMethod` , you are required to collect and include the account holder's name and the customer's email address.

## Data argument properties

**payment\_method** REQUIRED object

Pass an object to confirm using data collected without an `Element`.

**× Hide payment\_method properties****au\_becs\_debit** REQUIRED object**× Hide au\_becs\_debit properties****bsb\_number** REQUIRED string

A Bank State Branch (BSB) number.

**account\_number** REQUIRED string

A bank account number.

**billing\_details** REQUIRED Object

The customer's `billing_details` . `name` and `email` are required.

**× Hide billing\_details properties****name** REQUIRED string

The account holder's name.

**email** REQUIRED string

The customer's email.

## Confirm a BECS Debit payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmAuBecsDebitPayment`. It will be used to confirm the `PaymentIntent`.

### Data argument properties

---

`payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

## Confirm a BECS Debit payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmAuBecsDebitPayment` without passing in any additional data.

---

`stripe.confirmBancontactPayment(clientSecret, data?, options?)`

Use `stripe.confirmBancontactPayment` in the [Bancontact Payments with Payment Methods](#) flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

---

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[X Hide data properties](#)**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

**setup\_future\_usage** string

To set up a SEPA Direct Debit payment method using the bank details from this Bancontact payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the [PaymentIntent](#). The newly created SEPA Direct Debit [PaymentMethod](#) will be attached to this customer.

**options** optional object

An options object to control the behavior of this method.

[X Hide options properties](#)**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

**Returns**

`stripe.confirmBancontactPayment` will trigger a redirect when successful. If there is an error, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent` : the successful [PaymentIntent](#) .
- `result.error` : an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmBancontactPayment` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Bancontact payment with an existing payment method

Use `stripe.confirmBancontactPayment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

#### `setup_future_usage` string

To set up a SEPA Direct Debit payment method using the bank details from this Bancontact payment, set this parameter to `off_session`. When using this parameter, a `customer` will need

to be set on the `PaymentIntent`. The newly created SEPA Direct Debit `PaymentMethod` will be attached to this customer.

## Confirm a Bancontact payment with collected data

Your customer's name is required for the Bancontact authorization to succeed. You can pass in the customer's name directly to create a new `PaymentMethod` and confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm with the customer's name.

##### `x Hide payment_method properties`

#### `billing_details` REQUIRED object

An object detailing billing information.

##### `x Hide billing_details properties`

#### `name` REQUIRED string

The customer's name.

#### `email` string

The customer's email. Required when `setup_future_usage` is set to `off_session`.

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

---

#### `setup_future_usage` string

To set up a SEPA Direct Debit payment method using the bank details from this Bancontact payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the `PaymentIntent`. The newly created SEPA Direct Debit `PaymentMethod` will be attached to this customer.

---

## Confirm a Bancontact payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmBancontactPayment` without passing in any additional data.

---

### `stripe.confirmBlikPayment(clientSecret, data, options?)`

Use `stripe.confirmBlikPayment` in the **BLIK Payments with Payment Methods** flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically prompt the customer to authorize the transaction.

---

#### Method parameters

---

##### `clientSecret` REQUIRED string

The **client secret** of the `PaymentIntent`.

#### **data** REQUIRED object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

##### **payment\_method\_options** REQUIRED object

An object that contains transaction specific data.

[× Hide payment\\_method\\_options properties](#)

###### **code** REQUIRED string

Your customer's 6-digit BLIK code.

#### **payment\_method** object

Use this parameter to supply additional data relevant to the transaction, such as billing details.

[× Hide payment\\_method properties](#)

###### **billing\_details** object

The **billing details** associated with the transaction.

#### **options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)**handleActions** boolean

Set this to `false` if you want to manually determine if the confirmation has succeeded or failed.

## Returns

By default, `stripe.confirmBlikPayment` will only return when the payment has succeeded or failed. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmBlikPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## `stripe.confirmBoletoPayment(clientSecret, data?, options?)`

Use `stripe.confirmBoletoPayment` in the [Boleto Payment](#) with Payment Methods flow when the customer submits your payment form. When called, it will confirm the [PaymentIntent](#)

with `data` you provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

---

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

#### `payment_method` object | string

The `id` of an existing `PaymentMethod` or an object of collected data. See use cases below for details.

---

### `options` optional object

An options object to control the behavior of this method.

[× Hide options properties](#)**handleActions** REQUIRED boolean

Set to `false`. The Boleto private beta does not handle the next actions for you automatically (e.g. display Boleto details). Please refer to our [Stripe Boleto integration guide](#) for more info.

## Returns

`stripe.confirmBoletoPayment` will return a `Promise` which resolves with a result object. This object has either:

- `result.paymentIntent`: the successful PaymentIntent.
- `result.error`: an error. Refer to the API reference for all possible errors.

**i** Note that `stripe.confirmBoletoPayment` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Boleto payment with collected data

You can pass in the customer's billing details to create a new `PaymentMethod` and confirm the `PaymentIntent`. To create a Boleto `PaymentMethod`, you are required to collect and include the

customer's name, email, Brazilian tax id (CPF/CNPJ) and address.

## Data argument properties

---

### **payment\_method** REQUIRED object

Pass an object to confirm using data collected.

**× Hide payment\_method properties**

**boleto REQUIRED Object**

**× Hide boleto properties**

**tax\_id REQUIRED string**

The customer's Brazilian tax id (CPF/CNPJ).

**billing\_details REQUIRED Object**

The customer's **billing\_details** . `name` , `email` and `address` .

**× Hide billing\_details properties**

**name REQUIRED string**

The customer's name. The first and last name must be at minimum 2 characters each.

**email REQUIRED string**

The customer's email.

**address REQUIRED string**

The customer's address: street name, city, state and postal code

**× Hide address properties**

**line1 REQUIRED string**

The customer's address line 1 (e.g. street or company name).

**line2 string**

The customer's address line 2 (e.g. apartment, suite, unit, or building).

**city REQUIRED string**

The customer's address city (e.g. Sao Paulo).

**state** REQUIRED string

The customer's address state (e.g. SP).

**postal\_code** REQUIRED string

The customer's CEP (i.e. Brazilian postal code). Must be 8 digits long.

**country** REQUIRED string

Must be BR.

## Confirm a Boleto payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmBoletoPayment`. It will be used to confirm the `PaymentIntent`.

### Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod` .

## Confirm a Boleto payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmBoletoPayment` without passing in any additional data.

---

### `stripe.confirmCustomerBalancePayment(clientSecret, data?, options?)`

Use `stripe.confirmCustomerBalancePayment` in the **Customer Balance** payment flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide. Refer to our [integration guide](#) for more details.

Since the **Customer Balance** payment method draws from a balance, the attempt will succeed or fail depending on the current balance amount. To collect more funds from the customer when the cash balance is insufficient, use the customer balance with **bank transfer funding** parameters.

The confirmation attempt will finish in one of the following result states:

- If the customer balance is greater than or equal to the amount, the `PaymentIntent` response will have a `status` of `succeeded`. The `funding_type` will be ignored.
- If the customer balance is less than the amount, and you do not set the `funding_type`, the `PaymentIntent` response will have a `status` of `requires_payment_method`.
- If the customer balance is less than the amount, and you set the `funding_type`, the `PaymentIntent` response will have a `status` of `requires_action`. The `paymentIntent.next_action.display_bank_transfer_instructions` hash will contain bank transfer details for funding the **Customer Balance**.

---

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

**data** optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

**x Hide data properties****payment\_method** REQUIRED object

An object specifying the `customer_balance` type.

**x Hide payment\_method properties****customer\_balance** REQUIRED object

Set to `{}`.

**payment\_method\_options** object

Additional payment specific configuration options. See the [with collected data](#) use case below.

**options** optional object

An options object to control the behavior of this method.

**x Hide options properties****handleActions** REQUIRED boolean

Set to `false`. The [Customer Balance](#) does not handle the next actions for you automatically (e.g. displaying bank transfer details). To make future upgrades easier, this option is required to always be sent. Please refer to our [Stripe Customer Balance integration guide](#) for more info.

## Returns

`stripe.confirmCustomerBalancePayment` will return a `Promise` which resolves with a result object. This object has either:

- `result.paymentIntent` : when the **Customer Balance** is sufficient to pay the amount, a **PaymentIntent** with `status` of `succeeded`.
- `result.paymentIntent` : when the **Customer Balance** is insufficient to pay the amount, and `funding_type` data was provided, a **PaymentIntent** with `status` of `requires_action`.
- `result.paymentIntent` : when the **Customer Balance** is insufficient to pay the amount, and no `funding_type` data was provided, a **PaymentIntent** with `status` of `requires_payment_method`.
- `result.error` : an error. Refer to the API reference for all possible errors.

**i** Note that `stripe.confirmCustomerBalancePayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Customer Balance payment with collected data

You can pass in the customer's **bank transfer funding** details to create a new `PaymentMethod` and confirm the `PaymentIntent`.

If the **Customer Balance** was not enough to pay the amount, the `status` is `requires_action`. The `paymentIntent.next_action.display_bank_transfer_instructions` hash contains bank

transfer details for funding the [Customer Balance](#).

## Data argument properties

### **payment\_method** REQUIRED object

Pass an object to confirm using data collected.

[× Hide payment\\_method properties](#)

#### **customer\_balance** REQUIRED object

Set to `{}`.

### **payment\_method\_options** object

Additional payment-specific configuration options.

[× Hide payment\\_method\\_options properties](#)

#### **funding\_type** string

The funding method type to be used when there are not enough funds in the [Customer Balance](#). Permitted values include: `bank_transfer`.

#### **bank\_transfer** object

The customer's chosen bank transfer method.

[+ Show bank\\_transfer properties](#)

---

`stripe.confirmCashappPayment(clientSecret, data?, options?)`

Use `stripe.confirmCashappPayment` in the Cash App Pay payment method creation flow when the customer submits your payment form. When called, it will confirm the [PaymentIntent](#) with `data` you provide and handle the [NextAction](#) for the customer to authorize the payment.

When you confirm a [PaymentIntent](#), it needs to have an attached [PaymentMethod](#).

In addition to confirming the [PaymentIntent](#), this method can automatically create and attach a new [PaymentMethod](#) for you. If you have already attached a [PaymentMethod](#) you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The [client secret](#) of the [PaymentIntent](#).

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

#### [X Hide data properties](#)

##### `payment_method` RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

##### `return_url` REQUIRED string

The url your customer will be directed to after they complete authentication.

### options optional object

An options object to control the behavior of this method.

[X Hide options properties](#)

#### handleActions boolean

Set this to `false` if you would like to **handle displaying the Cash App Pay QR code or handle the authorization redirect** yourself.

## Returns

By default, `stripe.confirmCashappPayment` will display Cash App Pay QR code in desktop web app, or trigger a redirect in mobile web app. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent` : the successful **PaymentIntent** .
- `result.error` : an error. Refer to the **API reference** for all possible errors.

**i** Note that `stripe.confirmCashappPayment` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Cash App Pay payment Without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

---

`return_url` **REQUIRED** string

The url your customer will be directed to after they complete authentication.

---

## Confirm a Cash App Pay payment With an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

---

`payment_method` **REQUIRED** string

The `id` of an existing `PaymentMethod`.

---

`return_url` **REQUIRED** string

The url your customer will be directed to after they complete authentication.

---

## Confirm a Cash App Pay payment With an attached PaymentMethod

If you have already attached a `return_url` and a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

---

### `stripe.confirmEpsPayment(clientSecret, data?, options?)`

Use `stripe.confirmEpsPayment` in the **EPS Payments with Payment Methods** flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`. When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

`clientSecret` **REQUIRED** string

The `client secret` of the `PaymentIntent`.

#### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

##### `x Hide data properties`

###### `payment_method` RECOMMENDED string | object

Either the `id` of an existing `PaymentMethod`, or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

###### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

#### `options` optional object

An options object to control the behavior of this method.

##### `x Hide options properties`

###### `handleActions` boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

`stripe.confirmEpsPayment` will trigger a redirect when successful. If there is an error, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent` : the successful [PaymentIntent](#) .
- `result.error` : an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmEpsPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm an EPS payment with an existing payment method

Use `stripe.confirmEpsPayment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm an EPS payment with collected data

Your customer's name is required for the EPS authorization to succeed. You can pass in the customer's name directly to create a new `PaymentMethod` and confirm the `PaymentIntent`.

## Data argument properties

### `payment_method` REQUIRED object

Pass an object to confirm with the customer's name.

#### `× Hide payment_method properties`

##### `billing_details` REQUIRED object

An object detailing billing information.

#### `× Hide billing_details properties`

##### `name` REQUIRED string

The customer's name.

### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm an EPS payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmEpsPayment` without passing in any additional data.

---

### `stripe.confirmFpxPayment(clientSecret, data?, options?)`

Use `stripe.confirmFpxPayment` in the [FPX payment method creation](#) flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

#### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

#### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

**options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)

**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

By default, `stripe.confirmFpxPayment` will trigger a redirect when successful. If there is an error, or when handling `next_action`s manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.



Note that `stripe.confirmFpxPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm an FPX payment with payment data from an Element

Create and attach a new PaymentMethod by passing an `fpxBank` **Element** to `payment_method[fpx]`. The new `PaymentMethod` will be created with the **bank code** collected by the `Element` and will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an `object` to confirm using data collected by an `fpxBank` **Element**.

`x Hide payment_method properties`

`fpx` RECOMMENDED Element

An `fpxBank` **Element**.

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm an FPX payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

---

`payment_method` **REQUIRED** string

The `id` of an existing `PaymentMethod`.

---

`return_url` **RECOMMENDED** string

The url your customer will be directed to after they complete authentication.

---

## Confirm an FPX payment with self collected data

If you already know the customer's bank or want to collect it yourself, then you do not need to use the `fpxBank` **Element**. You can pass in the customer's **bank code** directly.

### Data argument properties

---

`payment_method` **REQUIRED** object

Pass an object to confirm using data collected by an `fpxBank` **Element**.

× Hide payment\_method properties

**fpx** REQUIRED object

An object detailing the customer's FPX bank.

× Hide fpx properties

**bank** REQUIRED string

The customer's **bank**.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm an FPX payment with an attached PaymentMethod

If you have already attached a `return_url` and a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmFpxPayment` without passing in any additional data.

### `stripe.confirmGiropayPayment(clientSecret, data?, options?)`

Use `stripe.confirmGiropayPayment` in the [giropay Payments with Payment Methods](#) flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the

transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`. When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

#### `x Hide data properties`

##### `payment_method` RECOMMENDED string | object

Either the `id` of an existing `PaymentMethod`, or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

##### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

### `options` optional object

An options object to control the behavior of this method.

[× Hide options properties](#)**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

`stripe.confirmGiropayPayment` will trigger a redirect when successful. If there is an error, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.



Note that `stripe.confirmGiropayPayment` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a giropay payment with an existing payment method

Use `stripe.confirmGiropayPayment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

## Data argument properties

---

### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

---

## Confirm a giropay payment with collected data

Your customer's name is required for the giropay authorization to succeed. You can pass in the customer's name directly to create a new `PaymentMethod` and confirm the `PaymentIntent`.

## Data argument properties

---

### `payment_method` REQUIRED object

Pass an object to confirm with the customer's name.

× Hide payment\_method properties

**billing\_details** REQUIRED object

An object detailing billing information.

× Hide billing\_details properties

**name** REQUIRED string

The customer's name.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm a giropay payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmGiropayPayment` without passing in any additional data.

### `stripe.confirmGiropayPayment(clientSecret, data?, options?)`

Use `stripe.confirmGiropayPayment` in the **GrabPay payments with Payment Methods** flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent`, and automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

## Method parameters

### clientSecret REQUIRED string

The **client secret** of the `PaymentIntent`.

### data optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

#### `x Hide data properties`

##### `payment_method` string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

##### `return_url` REQUIRED string

The URL your customer will be directed to after they complete authentication.

### options optional object

An options object to control the behavior of this method.

#### `x Hide options properties`

##### `handleActions` boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

By default, `stripe.confirmGrabPayPayment` will trigger a redirect when successful. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful **PaymentIntent**.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmGrabPayPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a GrabPay payment with a new PaymentMethod

You can confirm the `PaymentIntent` using `stripe.confirmGrabPayPayment` without passing in any additional data. This will automatically create and attach a new `PaymentMethod`.

### Data argument properties

`return_url` REQUIRED string

The URL your customer will be directed to after they complete authentication.

## Confirm a GrabPay payment with an existing PaymentMethod

Use `stripe.confirmGrabPayPayment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

---

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

#### `return_url` REQUIRED string

The URL your customer will be directed to after they complete authentication.

---

## Confirm a GrabPay payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, you can then confirm the `PaymentIntent` using `stripe.confirmGrabPayPayment` without passing in any additional data.

### Data argument properties

---

`return_url` **REQUIRED** string

The URL your customer will be directed to after they complete authentication.

---

### `stripe.confirmIdealPayment(clientSecret, data?, options?)`

Use `stripe.confirmIdealPayment` in the **iDEAL Payments with Payment Methods** flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

---

### Method parameters

**clientSecret** REQUIRED string

The **client secret** of the `PaymentIntent`.

**data** optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[X Hide data properties](#)**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

**setup\_future\_usage** string

To set up a SEPA Direct Debit payment method using the bank details from this iDEAL payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the [PaymentIntent](#). The newly created SEPA Direct Debit [PaymentMethod](#) will be attached to this customer.

**options** optional object

An options object to control the behavior of this method.

[X Hide options properties](#)**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

By default, `stripe.confirmIdealPayment` will trigger a redirect when successful. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmIdealPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm an iDEAL payment with payment data from an Element

Create and attach a new `PaymentMethod` with `stripe.confirmIdealPayment` by passing an `idealBank` [Element](#) to `payment_method[ideal]`. The new `PaymentMethod` will be created with the [bank code](#) collected by the `Element` and will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected by an `idealBank` [Element](#).

[X Hide payment\\_method properties](#)**ideal** RECOMMENDED Element

An `idealBank` Element.

**billing\_details** object

The customer's `billing_details`. Required when `setup_future_usage` is set to `off_session`.

[+ Show billing\\_details properties](#)**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

**setup\_future\_usage** string

To set up a SEPA Direct Debit payment method using the bank details from this iDEAL payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the `PaymentIntent`. The newly created SEPA Direct Debit `PaymentMethod` will be attached to this customer.

## Confirm an iDEAL payment with an existing payment method

Use `stripe.confirmIdealPayment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

## Data argument properties

### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

### `setup_future_usage` string

To set up a SEPA Direct Debit payment method using the bank details from this iDEAL payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the `PaymentIntent`. The newly created SEPA Direct Debit `PaymentMethod` will be attached to this customer.

## Confirm an iDEAL payment with self collected data

If you already know the customer's bank or want to collect it yourself, then you do not need to use the `idealBank` `Element`. You can pass in the customer's `bank code` directly to create a new `PaymentMethod` and confirm the `PaymentIntent`.

## Data argument properties

### `payment_method` REQUIRED object

Pass an object to confirm using data collected by an `idealBank` `Element`.

### [X Hide payment\\_method properties](#)

#### **ideal** REQUIRED object

An object detailing the customer's iDEAL bank.

##### [X Hide ideal properties](#)

#### **bank** REQUIRED string

The customer's **bank**.

#### **billing\_details** object

The customer's **billing\_details**. Required when `setup_future_usage` is set to `off_session`.

##### [+ Show billing\\_details properties](#)

#### **return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

#### **setup\_future\_usage** string

To set up a SEPA Direct Debit payment method using the bank details from this iDEAL payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the **PaymentIntent**. The newly created SEPA Direct Debit **PaymentMethod** will be attached to this customer.

## Confirm an iDEAL payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmIdealPayment` without passing in any additional data.

---

## `stripe.confirmKlarnaPayment(clientSecret, data?, options?)`

Use `stripe.confirmKlarnaPayment` in the Klarna payment method creation flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

#### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

#### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[X Hide data properties](#)

**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

**options** optional object

An options object to control the behavior of this method.

[X Hide options properties](#)

**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

By default, `stripe.confirmKlarnaPayment` will trigger a redirect when successful. If there is an error, or when handling `next_action`s manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.



Note that `stripe.confirmKlarnaPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Klarna payment Without an existing payment method

If you have not already created a `PaymentMethod`, the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

`return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm a Klarna payment With an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod`.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm a Klarna payment With an attached PaymentMethod

If you have already attached a `return_url` and a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

### `stripe.confirmKonbiniPayment(clientSecret, data?, options?)`

Use `stripe.confirmKonbiniPayment` in the **Konbini** payment flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### clientSecret REQUIRED string

The **client secret** of the `PaymentIntent`.

### data optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

#### `payment_method` object | string

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

#### `payment_method_options` Object

An object containing payment-method-specific configuration to confirm the `PaymentIntent` with.

[+ Show payment\\_method\\_options properties](#)

### options optional object

An options object to control the behavior of this method.

[× Hide options properties](#)

#### `handleActions` boolean

Set this to `false` if you want to handle next actions yourself. Please refer to our [integration guide](#) for more info. Default is `true`.

## Returns

`stripe.confirmKonbiniPayment` will return a `Promise` which resolves with a result object. This object has either:

- `result.paymentIntent`: the successful PaymentIntent.
- `result.error`: an error. Refer to the API reference for all possible errors.

**i** Note that `stripe.confirmKonbiniPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Konbini payment with collected data

You can pass in the customer's billing details to create a new `PaymentMethod` and confirm the `PaymentIntent`. To create a Konbini `PaymentMethod`, you are required to collect and include the customer's name and email.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected.

[× Hide payment\\_method properties](#)

**billing\_details** REQUIRED Object

The customer's **billing\_details**.

[× Hide billing\\_details properties](#)

**name** REQUIRED string

The customer's full name.

**email** REQUIRED string

The customer's email address.

**payment\_method\_options** Object

An object containing payment-method-specific configuration to confirm the **PaymentIntent** with.

[+ Show payment\\_method\\_options properties](#)

## Confirm a Konbini payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmKonbiniPayment`. It will be used to confirm the `PaymentIntent`.

### Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing [PaymentMethod](#).

---

**payment\_method\_options** Object

An object containing payment-method-specific configuration to confirm the [PaymentIntent](#) with.

+ Show `payment_method_options` properties

## Confirm a Konbini payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmKonbiniPayment` without passing in any additional data.

### Data argument properties

---

**payment\_method\_options** Object

An object containing payment-method-specific configuration to confirm the [PaymentIntent](#) with.

+ Show `payment_method_options` properties

---

`stripe.confirmOxxoPayment(clientSecret, data?, options?)`

Use `stripe.confirmOxxoPayment` in the [OXXO Payment](#) with Payment Methods flow when the customer submits your payment form. When called, it will confirm the [PaymentIntent](#) with `data` you provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `PaymentIntent`, it needs to have an attached [PaymentMethod](#). In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The [client secret](#) of the `PaymentIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

#### `payment_method` object | string

The `id` of an existing `PaymentMethod` or an object of collected data. See use cases below for details.

### `options` optional object

An options object to control the behavior of this method.

[× Hide options properties](#)**handleActions** boolean

Set this to `false` if you want to handle next actions yourself. Please refer to our [Stripe OXXO integration guide](#) for more info. Default is `true`.

## Returns

`stripe.confirmOxxoPayment` will return a `Promise` which resolves with a result object. This object has either:

- `result.paymentIntent`: the successful PaymentIntent.
- `result.error`: an error. Refer to the API reference for all possible errors.

**i** Note that `stripe.confirmOxxoPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.confirmOxxoPayment` will pop up a modal with the voucher. This modal contains all the information required to pay the voucher at OXXO stores, such as the amount, a reference number and corresponding barcode.

## Confirm an Oxxo payment with collected data

You can pass in the customer's billing details to create a new `PaymentMethod` and confirm the `PaymentIntent`. To create a OXXO `PaymentMethod`, you are required to collect and include the customer's name and email address.

## Data argument properties

### `payment_method` REQUIRED object

Pass an object to confirm using data collected.

#### `× Hide payment_method properties`

##### `billing_details` REQUIRED Object

The customer's `billing_details` . `name` and `email` are required.

#### `× Hide billing_details properties`

##### `name` REQUIRED string

The customer's name. The first and last name must be at minimum 2 characters each.

##### `email` REQUIRED string

The customer's email.

## Confirm an Oxxo payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmOxxoPayment`. It will be used to confirm the `PaymentIntent`.

## Data argument properties

---

`payment_method` **REQUIRED** string

The `id` of an existing **PaymentMethod**.

---

## Confirm an Oxxo payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmOxxoPayment` without passing in any additional data.

---

`stripe.confirmP24Payment(clientSecret, data?, options?)`

Use `stripe.confirmP24Payment` in the [Przelewy24 Payments with Payment Methods](#) flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

When you confirm a `PaymentIntent`, it needs to have an attached **PaymentMethod**. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### clientSecret REQUIRED string

The **client secret** of the `PaymentIntent`.

### data optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

#### `x Hide data properties`

##### `payment_method` RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

##### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

### options optional object

An options object to control the behavior of this method.

#### `x Hide options properties`

##### `handleActions` boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

`stripe.confirmP24Payment` will trigger a redirect when successful. If there is an error, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmP24Payment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Przelewy24 payment with an existing payment method

Use `stripe.confirmP24Payment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

---

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

---

## Confirm a Przelewy24 payment with collected data

Your customer's email address is required for the Przelewy24 authorization to succeed. You can pass in the customer's email address directly to create a new `PaymentMethod` and confirm the `PaymentIntent`.

### Data argument properties

---

#### `payment_method` REQUIRED object

Pass an object to confirm with the customer's email address.

× Hide payment\_method properties

**billing\_details** REQUIRED object

An object detailing billing information.

× Hide billing\_details properties

**email** REQUIRED string

The customer's email address.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm a Przelewy24 payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmP24Payment` without passing in any additional data.

### `stripe.confirmPayNowPayment(clientSecret, data?, options?)`

Use `stripe.confirmPayNowPayment` in the PayNow payment method creation flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with

`data` you provide and handle the [NextAction](#) for the customer to authorize the payment. When you confirm a `PaymentIntent`, it needs to have an attached [PaymentMethod](#). In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The [client secret](#) of the `PaymentIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

#### `x Hide data properties`

##### `payment_method` RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

### `options` optional object

An options object to control the behavior of this method.

#### `x Hide options properties`

##### `handleActions` boolean

Default is `true`. Set this to `false` if you would like to [handle displaying the PayNow QR code yourself](#).

## Returns

By default, `stripe.confirmPayNowPayment` will display the PayNow QR code overlay. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmPayNowPayment` may take several seconds to complete and display the QR code. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a PayNow payment Without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass payment method billing details.

[× Hide payment\\_method properties](#)

**billing\_details** RECOMMENDED object

The **billing\_details** associated with the payment method.

## Confirm a PayNow payment With an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod`.

## Confirm a PayNow payment With an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

## stripe.confirmPayPalPayment(clientSecret, data?)

Use `stripe.confirmPayPalPayment` in the [PayPal Payments with Payment Methods](#) flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent`, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

### Method parameters

#### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

#### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

##### `x Hide data properties`

#### `payment_method` string | object

The `id` of an existing [PaymentMethod](#). See the use case sections below for details.

#### `return_url` REQUIRED string

The url your customer will be directed to after they complete authentication.

### Returns

`stripe.confirmPayPalPayment` will trigger a redirect when successful. If there is an error, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent` : the successful [PaymentIntent](#) .
- `result.error` : an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmPayPalPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a PayPal payment with a new PaymentMethod

You can confirm the `PaymentIntent` using `stripe.confirmPayPalPayment` without passing in any additional data. This will automatically create and attach a new `PaymentMethod`.

### Data argument properties

`return_url` **REQUIRED** string

The url your customer will be directed to after they complete authentication.

## Confirm a PayPal payment with an existing payment method

Use `stripe.confirmPayPalPayment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

## Data argument properties

---

### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

### `return_url` REQUIRED string

The url your customer will be directed to after they complete authentication.

---

## Confirm a PayPal payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmPayPalPayment` without passing in any additional data.

---

### `stripe.confirmPixPayment(clientSecret, data?, options?)`

Use `stripe.confirmPixPayment` in the **Pix Payment** with Payment Methods flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

---

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this

method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### **clientSecret** REQUIRED string

The **client secret** of the `PaymentIntent`.

### **data** optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

#### **payment\_method** string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

### **options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)

#### **handleActions** boolean

Default to be `true`. Set this to `false` if you would like to [handle displaying the Pix QR code yourself](#).

## Returns

By default, `stripe.confirmPixPayment` will display the Pix QR code overlay. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful `PaymentIntent`.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmPixPayment` may take several seconds to complete and display the QR code. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Pix payment without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass payment method paramters or and empty payment method object.

## Confirm a Pix payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

---

`payment_method` **REQUIRED** string

The `id` of an existing `PaymentMethod`.

---

## Confirm a Pix payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

---

### `stripe.confirmPromptPayPayment(clientSecret, data?, options?)`

Use `stripe.confirmPromptPayPayment` in the PromptPay payment method creation flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide and handle the `NextAction` for the customer to authorize the payment. When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can

automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

#### `payment_method` RECOMMENDED string | object

Either the `id` of an existing `PaymentMethod`, or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

### `options` optional object

An options object to control the behavior of this method.

[× Hide options properties](#)

#### `handleActions` boolean

Default to be `true`. Set this to `false` if you would like to [handle displaying the PromptPay QR code yourself](#).

## Returns

By default, `stripe.confirmPromptPayPayment` will display the PromptPay QR code overlay. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful **PaymentIntent**.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmPromptPayPayment` may take several seconds to complete and display the QR code. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a PromptPay payment Without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass payment method billing details.

[× Hide payment\\_method properties](#)

**billing\_details** RECOMMENDED object

The **billing\_details** associated with the payment method.

## Confirm a PromptPay payment With an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod`.

## Confirm a PromptPay payment With an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

---

```
stripe.confirmSepaDebitPayment(clientSecret, data?)
```

Use `stripe.confirmSepaDebitPayment` in the [SEPA Direct Debit Payments](#) with Payment Methods flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

**clientSecret** REQUIRED string

The `client secret` of the `PaymentIntent`.

---

**data** optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[X Hide data properties](#)

**payment\_method** RECOMMENDED object | string

The `id` of an existing PaymentMethod or an object of collected data. See use cases below for details.

**setup\_future\_usage** string

To set up the SEPA Direct Debit account for reuse, set this parameter to `off_session`. SEPA Direct Debit only accepts an `off_session` value for this parameter. If a `customer` is provided on this [PaymentIntent](#), the [PaymentMethod](#) will be attached to the customer when the PaymentIntent transitions to `processing`.

## Returns

`stripe.confirmSepaDebitPayment` will return a `Promise` which resolves with a result object.

This object has either:

- `result.paymentIntent`: the successful PaymentIntent.
- `result.error`: an error. Refer to the API reference for all possible errors.

**i** Note that `stripe.confirmSepaDebitPayment` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a SEPA Debit payment with payment data from an Element

Create and attach a new PaymentMethod with `stripe.confirmSepaDebitPayment` by passing an `iban` `Element` to `payment_method[sepa_debit]`. The new `PaymentMethod` will be created with the data collected by the `Element` and will be used to confirm the `PaymentIntent`. Additionally, to create a SEPA Direct Debit `PaymentMethod`, you are required to collect and include the customer's name and email address.

### Data argument properties

---

#### `payment_method` REQUIRED object

Pass an `object` to confirm the payment using data collected by an `iban` `Element`.

### `<div>` Hide `payment_method` properties

#### `sepa_debit` REQUIRED Element

An `iban` Element.

#### `billing_details` REQUIRED Object

The customer's `billing_details` . `name` and `email` are required.

##### `<div>` Hide `billing_details` properties

###### `name` REQUIRED string

The customer's name.

###### `email` REQUIRED string

The customer's email.

## Confirm a SEPA Debit payment with self collected data

If you already know the customer's IBAN account number or want to collect it yourself, then you do not need to use the `iban` Element with `stripe.confirmSepaDebitPayment`. You can pass in the customer's account number directly to create a new `PaymentMethod` and confirm the `PaymentIntent`. To create a SEPA Direct Debit `PaymentMethod`, you are required to collect and include the customer's name and email address.

## Data argument properties

**payment\_method** REQUIRED object

Pass an object to confirm using data collected without an `Element`.

**x Hide payment\_method properties****sepa\_debit** REQUIRED object**x Hide sepa\_debit properties****iban** REQUIRED string

An IBAN account number.

**billing\_details** REQUIRED Object

The customer's `billing_details`. `name` and `email` are required.

**x Hide billing\_details properties****name** REQUIRED string

The customer's name.

**email** REQUIRED string

The customer's email.

## Confirm a SEPA Debit payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmSepaDebitPayment`. It will be used to confirm the `PaymentIntent`.

## Data argument properties

---

`payment_method` **REQUIRED** string

The `id` of an existing `PaymentMethod`.

---

## Confirm a SEPA Debit payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmSepaDebitPayment` without passing in any additional data.

---

`stripe.confirmSofortPayment(clientSecret, data?, options?)`

Use `stripe.confirmSofortPayment` in the **Sofort Payments with Payment Methods** flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide. It will then automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

#### `payment_method` RECOMMENDED string | object

Either the `id` of an existing `PaymentMethod`, or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

#### `setup_future_usage` string

To set up a SEPA Direct Debit payment method using the bank details from this SOFORT payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the `PaymentIntent`. The newly created SEPA Direct Debit `PaymentMethod` will be attached to this customer.

### `options` optional object

An options object to control the behavior of this method.

[X Hide options properties](#)

**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

`stripe.confirmSofortPayment` will trigger a redirect when successful. If there is an error, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

i Note that `stripe.confirmSofortPayment` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Sofort payment with an existing payment method

Use `stripe.confirmSofortPayment` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `PaymentIntent`.

## Data argument properties

### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

### `setup_future_usage` string

To set up a SEPA Direct Debit payment method using the bank details from this SOFORT payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the `PaymentIntent`. The newly created SEPA Direct Debit `PaymentMethod` will be attached to this customer.

## Confirm a Sofort payment with collected data

The country of your customer's bank is required for the Sofort authorization to succeed. You can pass in the country of your customer's bank directly to create a new `PaymentMethod` and confirm the `PaymentIntent`.

## Data argument properties

### `payment_method` REQUIRED object

Pass an object to confirm with the customer's name.

### `<!-- Hide payment_method properties -->`

#### **sofort** REQUIRED object

An object detailing SOFORT specific parameters.

##### `<!-- Hide sofort properties -->`

#### **country** REQUIRED string

The country code where customer's bank is located.

#### **billing\_details** object

The customer's **billing\_details**. Required when `setup_future_usage` is set to `off_session`.

##### `+ Show billing_details properties`

#### **return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

#### **setup\_future\_usage** string

To set up a SEPA Direct Debit payment method using the bank details from this SOFORT payment, set this parameter to `off_session`. When using this parameter, a `customer` will need to be set on the **PaymentIntent**. The newly created SEPA Direct Debit **PaymentMethod** will be attached to this customer.

## Confirm a Sofort payment with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm the `PaymentIntent` using `stripe.confirmSofortPayment` without passing in any additional data.

---

```
stripe.confirmSouthKoreaMarketPayment(  
  clientSecret,  
  data?,  
  options?  
)
```

Use `stripe.confirmSouthKoreaMarketPayment` in the South Korea Market payment method creation flow when the customer submits your payment form. When you call the method, it confirms the `PaymentIntent` with the `data` you provide, and automatically redirects the customer to authorize the transaction. Once authorization is complete, the customer is redirected back to your specified `return_url`. When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you've already attached a `return_url` and a `PaymentMethod`, you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

---

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)

**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

**options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)

**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

By default, `stripe.confirmSouthKoreaMarketPayment` triggers a redirect when successful. If there's an error, or when handling `next_action`s manually by using the `handleActions: false` option, it returns a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.



Note that `stripe.confirmSouthKoreaMarketPayment` might take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a South Korea Market payment without an existing payment method

If you haven't already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` is used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass payment method billing details.

##### [× Hide payment\\_method properties](#)

#### `billing_details` REQUIRED object

The `billing_details` associated with the payment.

##### [+ Show billing\\_details properties](#)

#### `shipping` RECOMMENDED object

The `shipping details` for the payment.

[+ Show shipping properties](#)**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

## Confirm a South Korea Market payment with an existing payment method

If you've already created a `PaymentMethod`, you can pass its `id` to `payment_method`, which is used to confirm the `PaymentIntent`.

### Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod`.

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

## Confirm a South Korea Market payment with an attached PaymentMethod

If you've already attached a `return_url` and a `PaymentMethod` to this `PaymentIntent`, you can confirm it without passing in any additional data.

---

### `stripe.confirmWechatPayPayment(clientSecret, data?, options?)`

Use `stripe.confirmWechatPayPayment` in the WeChat Pay payment method creation flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide and handle the `NextAction` for the customer to authorize the payment. When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

#### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

---

#### `data` optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[× Hide data properties](#)**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

**options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)**handleActions** boolean

Set this to `false` if you would like to [handle displaying the WeChat Pay QR code yourself](#).

## Returns

By default, `stripe.confirmWechatPayPayment` will display WeChat Pay QR code. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmWechatPayPayment` may take several seconds to complete and display the QR code. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a WeChat Pay payment Without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

---

#### `payment_method` REQUIRED object

Pass payment method billing details.

[× Hide payment\\_method properties](#)

#### `billing_details` RECOMMENDED object

The `billing_details` associated with the payment method.

## Confirm a WeChat Pay payment With an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

## Data argument properties

`payment_method` **REQUIRED** string

The `id` of an existing `PaymentMethod`.

## Confirm a WeChat Pay payment With an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

`stripe.confirmZipPayment(clientSecret, data?, options?)`

Use `stripe.confirmZipPayment` in the Zip payment method creation flow when the customer submits your payment form. When called, it will confirm the `PaymentIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

When you confirm a `PaymentIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `PaymentIntent`, this method can automatically create and attach a new `PaymentMethod` for you. If you have already attached a `return_url` and a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

**clientSecret** REQUIRED string

The **client secret** of the `PaymentIntent`.

**data** optional object

Data to be sent with the request. Refer to the [Payment Intents API](#) for a full list of parameters.

[X Hide data properties](#)**payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a `PaymentMethod` with. See the use case sections below for details.

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

**options** optional object

An options object to control the behavior of this method.

[X Hide options properties](#)**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

**Returns**

By default, `stripe.confirmZipPayment` will trigger a redirect when successful. If there is an error, or when handling `next_action`s manually by using the `handleActions: false` option, it will

return a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent` : the successful `PaymentIntent` .
- `result.error` : an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmZipPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm an Zip payment without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `PaymentIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass payment method billing details.

`x Hide payment_method properties`

#### `billing_details` REQUIRED object

The `billing_details` associated with the payment.

`+ Show billing_details properties`

**shipping** RECOMMENDED object

The **shipping details** for the payment.

+ Show shipping properties

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

## Confirm an Zip payment with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` and it will be used to confirm the `PaymentIntent`.

### Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod`.

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

## Confirm an Zip payment with an attached PaymentMethod

If you have already attached a `return_url` and a `PaymentMethod` to this `PaymentIntent`, then you can confirm without passing in any additional data.

---

### `stripe.retrievePaymentIntent(clientSecret)`

Retrieve a `PaymentIntent` using its `client secret`.

#### Method parameters

---

`clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent` to retrieve.

#### Returns

---

This method returns a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: a `PaymentIntent` was retrieved successfully.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

---

### `stripe.verifyMicrodepositsForPayment(clientSecret, data?)`

Use `stripe.verifyMicrodepositsForPayment` in the [Accept a Canadian pre-authorized debit payment](#) or [Accept an ACH Direct Debit payment](#) flow to verify a customer's bank account with micro-deposits.

It should be only called when `PaymentIntent` is in the `requires_action` state, and contains a `next_action` field that has a `type` equal to `verify_with_microdeposits`. Refer to our [integration guide](#) for more details.

## Method parameters

**clientSecret** REQUIRED string

The `client secret` of the `PaymentIntent`.

**data** optional object

Data to be sent with the request.

× Hide data properties

**amounts** array

An array of two positive integers, in *cents*, equal to the values of the micro-deposits sent to the bank account.

**descriptor\_code** string

A six-character code starting with SM present in the microdeposit sent to the bank account.

## Returns

`stripe.verifyMicrodepositsForPayment` will return a `Promise` which resolves with a result object. This object has either:

- `result.paymentIntent` : the **PaymentIntent** with a `status` of `processing`.
- `result.error` : an error. Refer to the [API reference](#) and our [integration guide](#) for all possible errors.

**i** Verification can fail for several reasons. The failure may happen synchronously as a direct error response, or asynchronously through a `payment_intent.payment_failed` webhook event. Refer to our [integration guide](#) for more details.

## stripe.createRadarSession()

Use `stripe.createRadarSession()` to associate client browser information with a Radar Session ID. This ID can then be passed to Stripe when [creating charges and payment methods](#) to associate client browser information with those charges and improve Radar's ability to identify fraud.

We've prefilled the example with a sample [test API key](#). Don't submit any personally identifiable information in requests made with this key. To create a Stripe object using your account, replace the sample API key with your actual API key or [sign in](#).

### Returns

This method returns a `Promise` which resolves with a `result` object. This object has either:

- `result.radarSession` : a Radar Session was created successfully.

- `result.error`: an error. Refer to the [API reference](#) for all possible errors and the [FPX guide](#) for FPX specific errors.

After you receive a Radar Session ID, pass it to your server and subsequently include it in your API requests to create charges, payment methods, or to create or confirm a PaymentIntent.

-  Note that `stripe.createRadarSession` should be non-blocking. If you receive an error from this function, continue on with completing charges without passing through a Radar Session ID.

## `stripe.handleNextAction(options)`

Use `stripe.handleNextAction` in the [finalizing payments on the server](#) flow to finish confirmation of a [PaymentIntent](#) with the `requires_action` status. It will throw an error if the PaymentIntent has a different status.

Depending on the payment method and required action, the customer may be temporarily redirected from your site and brought back to the `return_url` [parameter](#) provided when the PaymentIntent is confirmed.

### Method parameters

#### `options` REQUIRED object

 Hide options properties

`clientSecret` string

The [client secret](#) of the `PaymentIntent`.

## Returns

This method returns a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent` : a [PaymentIntent](#) with the `processing` or `succeeded` status.
- `result.error` : an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.handleNextAction` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.handleNextAction` may trigger a [3D Secure](#) authentication challenge. The authentication challenge requires a context switch that can be hard to follow on a screen-reader. Ensure that your form is accessible by ensuring that success or error messages are clearly read out.

## `stripe.handleCardAction(clientSecret)`

Use `stripe.handleCardAction` in the Payment Intents API [manual confirmation](#) flow to handle a [PaymentIntent](#) with the `requires_action` status. It will throw an error if the `PaymentIntent` has a different status.

### Method parameters

`clientSecret` REQUIRED string

The [client secret](#) of the [PaymentIntent](#) to handle.

## Returns

This method returns a [Promise](#) which resolves with a [result](#) object. This object has either:

- `result.paymentIntent`: a [PaymentIntent](#) with the `requires_confirmation` status to confirm server-side.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.handleCardAction` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.handleCardAction` may trigger a [3D Secure](#) authentication challenge. The authentication challenge requires a context switch that can be hard to follow on a screen-reader. Ensure that your form is accessible by ensuring that success or error messages are clearly read out.

## `stripe.collectBankAccountForPayment(options)`

Use `stripe.collectBankAccountForPayment` in the [Accept a payment](#) flow for the [ACH Direct Debit](#) payment method to collect the customer's bank account in your payment form. When called, it will automatically load an on-page modal UI to collect bank account details and verification, and attach the [PaymentMethod](#) to the [PaymentIntent](#).

## Method parameters

### options REQUIRED object

Data to be sent with the request.

#### × Hide options properties

##### clientSecret REQUIRED string

The **client secret** of the `PaymentIntent`.

##### params REQUIRED object

#### × Hide params properties

##### payment\_method\_type REQUIRED string

The payment method type for the bank account details (e.g. `us_bank_account`)

##### payment\_method\_data REQUIRED object

Payment method specific data to be sent with the request

+ Show payment\_method\_data properties

## Returns

When the `stripe.collectBankAccountForPayment` completes successfully, it returns a `PaymentIntent`. If the customer provided their account, the `PaymentIntent` is in the `requires_confirmation` state. If the customer closed the dialog without providing their account, the `PaymentIntent` is in the `requires_payment_method` state. Use `stripe.confirmUsBankAccountPayment` to complete the process.

## Setup Intents

Use the [Setup Intents APIs](#) to save a card and charge it later. For step-by-step instructions on using the Setup Intents APIs, see the [set up recurring payments guide](#).

The following Stripe.js methods are available for working with Setup Intents.

---

### `stripe.confirmSetup(options)`

Use `stripe.confirmSetup` to confirm a [SetupIntent](#) using data collected by the [Payment Element](#), or with manually provided data via `confirmParams`. When called, `stripe.confirmSetup` will attempt to complete any [required actions](#), such as authenticating your user by displaying a 3DS dialog or redirecting them to a bank authorization page. Your user will be redirected to the `return_url` you pass once the authorization is complete.

#### Method parameters

---

**options** REQUIRED object

[× Hide options properties](#)**elements** CONDITIONALLY REQUIRED object

The [Elements](#) instance used to create the Payment Element.

Required if you [collect payment details before creating an Intent](#). It's always required if you don't provide a `clientSecret`.

**clientSecret** CONDITIONALLY REQUIRED string

The SetupIntent's client secret.

Required if you [collect payment details before creating an Intent](#). It's always required if you don't provide an `elements` instance containing a [client secret](#).

**confirmParams** object

Parameters that will be passed on to the Stripe API. Refer to the [Setup Intents API](#) for a full list of parameters.

[× Hide confirmParams properties](#)**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

**payment\_method\_data** object

When you call `stripe.confirmSetup`, payment details are collected from the Element and passed to the SetupIntents [confirm endpoint](#) as the `payment_method_data` parameter. You can also include additional `payment_method_data` fields, which will be merged with the data collected from the Element.

[+ Show payment\\_method\\_data properties](#)**expand** array

An array of pass through [SetupIntent](#) expansion parameters ([learn more](#)).

```
redirect 'always' | 'if_required'
```

By default, `stripe.confirmSetup` will always redirect to your `return_url` after a successful confirmation. If you set `redirect: "if_required"`, then `stripe.confirmSetup` will only redirect if your user chooses a redirect-based payment method.

**Note:** Setting `if_required` requires that you handle successful confirmations for redirect-based and non-redirect based payment methods separately. When a non-redirect based payment method is successfully confirmed, `stripe.confirmSetup` will resolve with a `{setupIntent}` object.

## Returns

`stripe.confirmSetup` will return a `Promise`. Upon a successful authorization, your user will be redirected to the `return_url` you provide before the Promise ever resolves.

If the authorization fails, the `Promise` will resolve with an `{error}` object that describes the failure. When the `error type` is `card_error` or `validation_error`, you can display the error message in `error.message` directly to your user. An error type of `invalid_request_error` could be due to an invalid request or 3DS authentication failures.

Note that for some payment methods such as iDEAL or Afterpay Clearpay, your user will first be redirected to an intermediate page to authorize the payment. If they fail to authorize the payment, they will be redirected back to your `return_url` and the `SetupIntent` will have a `status` of `requires_payment_method`. In this case you should attempt to recollect payment from the user.

- i Note that `stripe.confirmSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting

indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a SetupIntent by payment method

Below are a number of methods used to confirm a SetupIntent for a specific payment method type.

### `stripe.confirmCardSetup(clientSecret, data?, options?)`

Use `stripe.confirmCardSetup` in the [Setup Intents API flow](#) when the customer submits your payment form. When called, it will confirm the `SetupIntent` with `data` you provide and carry out 3DS or other next actions if they are required.

When you confirm a `SetupIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `SetupIntent`, this method can automatically create and attach a new `PaymentMethod` for you. It can also be called with an existing `PaymentMethod`, or if you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

`clientSecret` **REQUIRED** string

The [client secret](#) of the [SetupIntent](#).

#### **data** optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

[× Hide data properties](#)

##### **payment\_method** RECOMMENDED string | object

Either the `id` of an existing [PaymentMethod](#), or an object containing data to create a [PaymentMethod](#) with. See the use case sections below for details.

##### **return\_url** string

If you are [handling next actions yourself](#), pass in a `return_url`. If the subsequent action is `redirect_to_url`, this URL will be used on the return path for the redirect.

#### **options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)

##### **handleActions** boolean

Set this to `false` if you want to [handle next actions yourself](#), or if you want to defer next action handling until later (e.g. for use in the [PaymentRequest API](#)). Default is `true`.

## Returns

`stripe.confirmCardSetup` will return a `Promise` which resolves with a `result` object. This object has either:

- `result.setupIntent`: the successful **SetupIntent**.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmCardSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.confirmCardSetup` may trigger a **3D Secure** authentication challenge. This will be shown in a modal dialog and may be confusing for customers using assistive technologies like screen readers. You should make your form accessible by ensuring that success or error messages are clearly read out after this method completes.

## Confirm card setup with payment data from an Element

Use `stripe.confirmCardSetup` with payment data from an **Element** by passing a `card` or `cardNumber` `Element` to `payment_method[card]`. The new `PaymentMethod` will be created with data collected by the `Element` and will be used to confirm the `SetupIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected by a `card` or `cardNumber` `Element`.

### `x Hide payment_method properties`

#### `card` REQUIRED Element

Uses the provided `card` or `cardNumber` Element for confirmation.

#### `billing_details` RECOMMENDED object

The `billing_details` associated with the card.

## Confirm card setup with an existing payment method

Use `stripe.confirmCardSetup` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `SetupIntent`.

### Data argument properties

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

## Confirm card setup with an existing token

For backwards compatibility, you can convert an existing `Token` into a `PaymentMethod` with `stripe.confirmCardSetup` by passing the `Token` to `payment_method[card][token]`. The newly

created `PaymentMethod` will be used to confirm the `PaymentMethod`.

## Data argument properties

### `payment_method` REQUIRED object

Pass an object to confirm using an existing token.

#### `× Hide payment_method properties`

##### `card` REQUIRED object

An object of card data.

#### `× Hide card properties`

##### `token` REQUIRED string

Converts the provided token into a `PaymentMethod` to use for confirmation.

##### `billing_details` RECOMMENDED object

The `billing_details` associated with the card.

## Confirm card setup with an attached `PaymentMethod`

If you have already attached a `PaymentMethod` to this `SetupIntent`, then you can confirm the `SetupIntent` using `stripe.confirmCardSetup` without passing in any additional data.

## stripe.confirmUsBankAccountSetup(clientSecret, data?)

Use `stripe.confirmUsBankAccountSetup` in the [Save bank details](#) flow for the [ACH Direct Debit](#) payment method to record the customer's authorization for future payments.

When you confirm a [SetupIntent](#), it needs to have an attached [PaymentMethod](#). We suggest using `stripe.collectBankAccountForSetup`, which automatically collects bank account details and attaches a [PaymentMethod](#). You may also choose to reuse an existing [PaymentMethod](#) or manually collect bank account details using the `data` parameter. These use cases are detailed in the sections that follow.

### Method parameters

#### `clientSecret` REQUIRED string

The [client secret](#) of the [SetupIntent](#).

#### `data` optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

##### [X Hide data properties](#)

###### `payment_method` object | string

The `id` of an existing [PaymentMethod](#) or an object of collected data. See use cases below for details.

### Returns

`stripe.confirmUsBankAccountSetup` will return a `Promise` which resolves with a result object.

This object has either:

- `result.setupIntent`: the successful **SetupIntent** .
  - `result.error`: an error. Refer to the [API reference](#) and our [integration guide](#) for all possible errors.
- 

## Confirm an ACH Direct Debit setup with an existing PaymentMethod

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmUsBankAccountSetup`.

### Data argument properties

---

`payment_method` **REQUIRED** string

The `id` of an existing **PaymentMethod** .

---

## Confirm an ACH Direct Debit setup with an attached PaymentMethod

If you have successfully called `stripe.collectBankAccountForSetup` or attached a `PaymentMethod` to this `SetupIntent` already, then you can confirm the `SetupIntent` without

passing in any additional data.

---

## Confirm an ACH Direct Debit setup with self collected bank account information

If you already know the customer's bank account information, or want to collect it yourself, you can pass them in directly to create a new `PaymentMethod` and confirm the `SetupIntent`.

### Data argument properties

---

`payment_method` **REQUIRED** object

Pass an object to confirm using data collected.

**x Hide payment\_method properties**

**billing\_details REQUIRED Object**

The customer's **billing\_details** . `name` is required. Providing `email` allows your customer to receive [ACH Direct Debit mandate and microdeposit emails](#).

**x Hide billing\_details properties**

**name REQUIRED string**

The customer's name. The first and last name must be at minimum 2 characters each.

**email string**

The customer's email.

**us\_bank\_account REQUIRED Object**

The customer's [bank account information](#) .

**x Hide us\_bank\_account properties**

**account\_number REQUIRED string**

The customer's bank account number.

**routing\_number REQUIRED string**

The routing number of the customer's bank.

**account\_holder\_type REQUIRED string**

Account holder type: individual or company.

**account\_type string**

Account type: checkings or savings. Defaults to checking if omitted.

---

## stripe.confirmAcssDebitSetup(`clientSecret`, `data?`, `options?`)

Use `stripe.confirmAcssDebitSetup` in the [Save bank details](#) flow to set up a [Canadian pre-authorized debit](#) payment method for future payments. When called, it will automatically pop up a modal to collect bank account details and verification, accept the mandate, and confirm the [SetupIntent](#) when the user submits the form. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `SetupIntent`, it needs to have an attached [PaymentMethod](#).

`stripe.confirmAcssDebitSetup` automatically creates a new `PaymentMethod` for you when your customer completes the modal UI. It can also be called with an existing `PaymentMethod`, which will load the modal UI to collect a new mandate agreement. These use cases are detailed in the sections that follow.

### Method parameters

---

#### `clientSecret` REQUIRED string

The [client secret](#) of the `SetupIntent`.

---

#### `data` optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

[× Hide data properties](#)**payment\_method** object | string

The `id` of an existing PaymentMethod or an object of collected data. See use cases below for details.

**options** optional object

An options object to control the behavior of this method.

[× Hide options properties](#)**skipMandate** boolean

Set this to `true` if you want to skip displaying the mandate confirmation.

## Returns

`stripe.confirmAcssDebitSetup` will return a `Promise` which resolves with a result object. This object has either:

- `result.setupIntent`: the successful **SetupIntent**.
- `result.error`: an error. Refer to the [API reference](#) and our [integration guide](#) for all possible errors.

**i** Note that `stripe.confirmAcssDebitSetup` may take several seconds to complete.

During that time, disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, show it to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a Canadian pre-authorized debit setup with a new PaymentMethod

You can pass in the customer's billing details to create a new `PaymentMethod` and confirm the `SetupIntent`. You are required to collect and include the customer's name and email address. This method loads an on-page modal UI that handles bank account details collection and verification, presents a hosted mandate agreement and collects authorization for you.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected.

##### `× Hide payment_method properties`

##### `billing_details` REQUIRED Object

The customer's `billing_details` . `name` and `email` are required.

##### `× Hide billing_details properties`

##### `name` REQUIRED string

The customer's name. The first and last name must be at minimum 2 characters each.

##### `email` REQUIRED string

The customer's email.

## Confirm a Canadian pre-authorized debit setup with an existing PaymentMethod

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmAcssDebitSetup`. This method loads an on-page modal UI that only presents a hosted mandate agreement and collects authorization for you.

### Data argument properties

---

`payment_method` REQUIRED string

The `id` of an existing `PaymentMethod` .

---

## Confirm a Canadian pre-authorized debit setup with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `SetupIntent`, then you can confirm the `SetupIntent` without passing in any additional data. This method loads an on-page modal UI that only presents a hosted mandate agreement and collects authorization for you.

---

## Confirm a Canadian pre-authorized debit setup with self collected bank account information

If you already know the customer's bank account information, or want to collect it yourself, you can pass them in directly to create a new `PaymentMethod` and confirm the `SetupIntent`. In this case, this method does not load the on-page modal UI, so you will need to [build your own mandate agreement page](#).

### Data argument properties

---

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected.

Hide payment\_method properties

**billing\_details** REQUIRED Object

The customer's **billing\_details** . `name` and `email` are required.

Hide billing\_details properties

**name** REQUIRED string

The customer's name. The first and last name must be at minimum 2 characters each.

**email** REQUIRED string

The customer's email.

**acss\_debit** REQUIRED Object

The customer's **bank account information** .

Hide acss\_debit properties

**account\_number** REQUIRED string

The customer's bank account number.

**institution\_number** REQUIRED string

The institution number of the customer's bank.

**transit\_number** REQUIRED string

The transit number of the customer's bank.

## Confirm a Canadian pre-authorized debit setup with an existing PaymentMethod but skip mandate display

If you have already created a `PaymentMethod` and built your own mandate agreement page, you can reuse it by passing its `id` to `payment_method` when calling `stripe.confirmAcssDebitSetup` and skip the on-page modal UI at the same time.

### Data and options argument parameters

#### `data` object

Data to be sent with the request.

[× Hide data properties](#)

`payment_method` **REQUIRED** string

The `id` of an existing `PaymentMethod`.

#### `options` object

An options object to control the behavior of this method.

[× Hide options properties](#)

`skipMandate` **REQUIRED** boolean

Set to `true` to skip the on-page modal UI.

## stripe.confirmAuBecsDebitSetup(`clientSecret`, `data?`)

Use `stripe.confirmAuBecsDebitSetup` in the [BECS Direct Debit Payments](#) flow when the customer submits your payment form. When called, it will confirm the [SetupIntent](#) with `data` you provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `SetupIntent`, it needs to have an attached [PaymentMethod](#). In addition to confirming the `SetupIntent`, this method can automatically create and attach a new [PaymentMethod](#) for you. It can also be called with an existing [PaymentMethod](#), or if you have already attached a [PaymentMethod](#) you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

#### `clientSecret` REQUIRED string

The [client secret](#) of the [SetupIntent](#).

#### `data` optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

##### [× Hide data properties](#)

#### `payment_method` RECOMMENDED object | string

The `id` of an existing [PaymentMethod](#) or an object of collected data. See use cases below for details.

### Returns

`stripe.confirmAuBecsDebitSetup` will return a `Promise` which resolves with a `result` object.

This object has either:

- `result.setupIntent`: the successful `SetupIntent`.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmAuBecsDebitSetup` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm BECS Debit setup with payment data from an Element

Create and attach a new `PaymentMethod` with `stripe.confirmAuBecsDebitSetup` by passing an `auBankAccount` `Element` to `payment_method[au_becs_debit]`. The new `PaymentMethod` will be created with the data collected by the `Element` and will be used to confirm the `SetupIntent`.

Additionally, to create a BECS Direct Debit `PaymentMethod`, you are required to collect and include the account holder's name and the customer's email address.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an `object` to confirm using data collected by an `auBankAccount` `Element`.

**x Hide payment\_method properties**

**au\_becs\_debit REQUIRED Element**

An `auBankAccount` **Element**.

**billing\_details REQUIRED Object**

The customer's **billing\_details**. `name` and `email` are required.

**x Hide billing\_details properties**

**name REQUIRED string**

The customer's name.

**email REQUIRED string**

The customer's email.

## Confirm BECS Debit setup with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmAuBecsDebitSetup()` and it will be used to confirm the `SetupIntent`.

### Data argument properties

---

`payment_method` **REQUIRED** string

The `id` of an existing `PaymentMethod`.

---

## Confirm BECS Debit setup with self collected data

If you already know the customer's BSB number and bank account number or want to collect it yourself, then you do not need to use the `auBankAccount` `Element`. You can pass in the customer's bank account information directly to create a new `PaymentMethod` and confirm the `SetupIntent`. Additionally, to create a BECS Direct Debit `PaymentMethod`, you are required to collect and include the account holder's name and the customer's email address.

### Data argument properties

---

`payment_method` **REQUIRED** object

Pass an object to confirm using data collected without an `Element`.

× Hide payment\_method properties

**au\_becs\_debit** REQUIRED object

An object of self-collected bank account data.

× Hide au\_becs\_debit properties

**bsb\_number** REQUIRED string

A Bank State Branch (BSB) number.

**account\_number** REQUIRED string

A bank account number.

**billing\_details** REQUIRED object

The customer's **billing\_details** . `name` and `email` are required.

× Hide billing\_details properties

**name** REQUIRED string

The account holder's name.

**email** REQUIRED string

The customer's email.

## `stripe.confirmBacsDebitSetup(clientSecret, data?)`

Use `stripe.confirmBacsDebitSetup` in the **Bacs Direct Debit Payments** flow when the customer submits your payment form. When called, it will confirm the `SetupIntent` with `data` you

provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `SetupIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `SetupIntent`, this method can automatically create and attach a new `PaymentMethod` for you. It can also be called with an existing `PaymentMethod`, or if you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `SetupIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

#### `x Hide data properties`

##### `payment_method` RECOMMENDED object | string

The `id` of an existing `PaymentMethod` or an object of collected data. See use cases below for details.

## Returns

`stripe.confirmBacsDebitSetup` will return a `Promise` which resolves with a `result` object.

This object has either:

- `result.setupIntent` : the successful `SetupIntent`.

- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmBacsDebitSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm Bacs Debit setup with self collected data

You can also pass in the customer's bank account information directly to create a new `PaymentMethod` and confirm the `SetupIntent`. Additionally, to create a Bacs Direct Debit `PaymentMethod`, you are required to collect and include the account holder's name and the customer's email address.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm using payment method data.

Hide payment\_method properties

**bacs\_debit** REQUIRED object

An object of self-collected bank account data.

Hide bacs\_debit properties

**account\_number** REQUIRED string

A bank account number.

**sort\_code** REQUIRED string

A sort code.

**billing\_details** REQUIRED object

The customer's **billing\_details** . `name` and `email` are required.

Hide billing\_details properties

**address** REQUIRED string

The account holder's address.

**name** REQUIRED string

The account holder's name.

**email** REQUIRED string

The customer's email.

## Confirm Bacs Debit setup with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmBacsDebitSetup` and it will be used to confirm the `SetupIntent`.

### Data argument properties

---

`payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

### `stripe.confirmBancontactSetup(clientSecret, data?, options?)`

Use `stripe.confirmBancontactSetup` in the [Set up future payments](#) flow to use Bancontact bank details to set up a SEPA Direct Debit payment method for future payments. When called, it will confirm a `SetupIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `SetupIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `SetupIntent`, this method can automatically create and attach a new `PaymentMethod` for you. It can also be called with an existing `PaymentMethod`, or if you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### clientSecret REQUIRED string

The **client secret** of the `SetupIntent`.

### data optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

#### `x Hide data properties`

##### `payment_method` RECOMMENDED object | string

The `id` of an existing PaymentMethod or an object of collected data. See use cases below for details.

##### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

### options optional object

An options object to control the behavior of this method.

#### `x Hide options properties`

##### `handleActions` boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

`stripe.confirmBancontactSetup` will return a `Promise` which resolves with a `result` object.

This object has either:

- `result.setupIntent`: the successful `SetupIntent`.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmBancontactSetup` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm Bancontact setup with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmBancontactSetup` and it will be used to confirm the `SetupIntent`.

### Data argument properties

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm Bancontact setup with self collected data

Your customer's name and email are required for the Bancontact authorization to succeed. You can pass in these properties directly to create a new `PaymentMethod` and confirm the `SetupIntent`.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm with the customer's name and email.

##### `x Hide payment_method properties`

##### `billing_details` REQUIRED object

The customer's `billing_details` . `name` and `email` are required.

##### `x Hide billing_details properties`

##### `name` REQUIRED string

The customer's name.

##### `email` REQUIRED string

The customer's email.

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## stripe.confirmCashappSetup(`clientSecret`, `data?`, `options?`)

Use `stripe.confirmCashappSetup` in the [Save payment details](#) flow for the [Cash App Pay](#) payment method to record the customer's authorization for future payments.

When you confirm a [SetupIntent](#), it needs to have an attached [PaymentMethod](#).

In addition to confirming the [SetupIntent](#), this method can automatically create and attach a new [PaymentMethod](#) for you. If you have already attached a [PaymentMethod](#) you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

#### `clientSecret` REQUIRED string

The [client secret](#) of the [SetupIntent](#).

#### `data` optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

##### [× Hide data properties](#)

###### `payment_method` object | string

The `id` of an existing PaymentMethod or an object of collected data. See use cases below for details.

###### `return_url` REQUIRED string

The url your customer will be directed to after they complete authentication.

#### `options` optional object

An options object to control the behavior of this method.

[X Hide options properties](#)

**handleActions** boolean

Set this to `false` if you would like to [handle displaying the Cash App Pay QR code or handle the authorization redirect](#) yourself.

## Returns

By default, `stripe.confirmCashappSetup` will display Cash App Pay QR code in desktop web app, or trigger a redirect in mobile web app. If there is an error, or when handling next actions manually by using the `handleActions: false` option, it will return a `Promise` which resolves with a `result`. This object has either:

- `result.setupIntent`: the successful [SetupIntent](#).
- `result.error`: an error. Refer to the [API reference](#) and our [integration guide](#) for all possible errors.

**i** Note that `stripe.confirmCashappSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm an Cash App Pay setup Without an existing payment method

If you have not already created a `PaymentMethod`, you can pass payment method parameters, and the newly created `PaymentMethod` will be used to confirm the `SetupIntent`.

### Data argument properties

---

`return_url` **REQUIRED** string

The url your customer will be directed to after they complete authentication.

---

## Confirm an Cash App Pay setup with an existing PaymentMethod

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmCashappSetup`.

### Data argument properties

---

`payment_method` **REQUIRED** string

The `id` of an existing `PaymentMethod`.

---

`return_url` **REQUIRED** string

The url your customer will be directed to after they complete authentication.

---

## Confirm an Cash App Pay setup with an attached PaymentMethod

If you have already attached a `return_url` and a `PaymentMethod` to this `SetupIntent`, then you can confirm without passing in any additional data.

---

### `stripe.confirmIdealSetup(clientSecret, data?, options?)`

Use `stripe.confirmIdealSetup` in the [Set up future payments](#) flow to use iDEAL bank details to set up a SEPA Direct Debit payment method for future payments. When called, it will confirm a `SetupIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected back to your specified `return_url`. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `SetupIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `SetupIntent`, this method can automatically create and attach a new `PaymentMethod` for you. It can also be called with an existing `PaymentMethod`, or if you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

### Method parameters

---

**clientSecret** REQUIRED string

The **client secret** of the `SetupIntent`.

**data** optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

**x Hide data properties****payment\_method** RECOMMENDED object | string

The `id` of an existing PaymentMethod or an object of collected data. See use cases below for details.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

**options** optional object

An options object to control the behavior of this method.

**x Hide options properties****handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

**Returns**

`stripe.confirmIdealSetup` will return a `Promise` which resolves with a `result` object. This object has either:

- `result.setupIntent`: the successful **SetupIntent**.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmIdealSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm iDEAL setup with payment data from an Element

Create and attach a new SEPA Direct Debit `PaymentMethod` with `stripe.confirmIdealSetup` by passing an `idealBank` **Element** to `payment_method[ideal]`. The new `PaymentMethod` will be created with the data collected by the `Element` and will be used to confirm the `SetupIntent`. Additionally, to create a SEPA Direct Debit `PaymentMethod`, you are required to collect and include the customer's name and email address.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an object to confirm using data collected by an `idealBank` Element.

### `<!-- Hide payment_method properties -->`

#### `ideal` REQUIRED Element

An `idealBank` Element.

#### `billing_details` REQUIRED object

The customer's `billing_details`. `.name` and `.email` are required.

##### `<!-- Hide billing_details properties -->`

###### `name` REQUIRED string

The customer's name.

###### `email` REQUIRED string

The customer's email.

#### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm iDEAL setup with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmIdealSetup` and it will be used to confirm the `SetupIntent`.

## Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod`.

**return\_url** RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## Confirm iDEAL setup with self collected data

If you already know the customer's bank or want to collect it yourself, then you do not need to use the `idealBank` Element. You can pass in the customer's `bank code` directly to create a new `PaymentMethod` and confirm the `SetupIntent`.

### Data argument properties

**payment\_method** REQUIRED object

Pass an object to confirm using data collected by an `idealBank` Element.

### `<!-- Hide payment_method properties -->`

#### `ideal` REQUIRED object

An object detailing the customer's iDEAL bank.

##### `<!-- Hide ideal properties -->`

###### `bank` REQUIRED string

The customer's `bank`.

### `billing_details` REQUIRED object

The customer's `billing_details`. `.name` and `email` are required.

##### `<!-- Hide billing_details properties -->`

###### `name` REQUIRED string

The customer's name.

###### `email` REQUIRED string

The customer's email.

### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

## `stripe.confirmPayPalSetup(clientSecret, data?)`

Use `stripe.confirmPayPalSetup` in the **PayPal Payments with Setup Intents** flow when the customer submits your setup form. When called, it will confirm the `SetupIntent`, and it will

automatically redirect the customer to authorize the setup. Once authorization is complete, the customer will be redirected back to your specified `return_url`.

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `SetupIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

#### `x Hide data properties`

##### `payment_method` string | object

The `id` of an existing `PaymentMethod`. See the use case sections below for details.

##### `return_url` REQUIRED string

The url your customer will be directed to after they complete authentication.

## Returns

`stripe.confirmPayPalSetup` will trigger a redirect when successful. If there is an error, it will return a `Promise` which resolves with a `result` object. This object has either:

- `result.setupIntent`: the successful `SetupIntent`.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.



Note that `stripe.confirmPayPalSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm a PayPal setup with a new PaymentMethod

You can confirm the `SetupIntent` using `stripe.confirmPayPalSetup` without passing in any additional data. This will automatically create and attach a new `PaymentMethod`.

### Data argument properties

`return_url` REQUIRED string

The url your customer will be directed to after they complete authentication.

## Confirm a PayPal setup with an existing payment method

Use `stripe.confirmPayPalSetup` with an existing `PaymentMethod` by passing its `id` to `payment_method`. The `PaymentMethod` will be used to confirm the `SetupIntent`.

### Data argument properties

**payment\_method** REQUIRED string

The `id` of an existing `PaymentMethod`.

**return\_url** REQUIRED string

The url your customer will be directed to after they complete authentication.

## Confirm a PayPal setup with an attached PaymentMethod

If you have already attached a `PaymentMethod` to this `SetupIntent`, then you can confirm the `SetupIntent` using `stripe.confirmPayPalSetup` without passing in any additional data.

### `stripe.confirmSepaDebitSetup(clientSecret, data?)`

Use `stripe.confirmSepaDebitSetup` in the [SEPA Direct Debit with Setup Intents](#) flow when the customer submits your payment form. When called, it will confirm the `SetupIntent` with `data` you provide. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `SetupIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `SetupIntent`, this method can automatically create and attach a new `PaymentMethod` for you. It can also be called with an existing `PaymentMethod`, or if you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### clientSecret REQUIRED string

The **client secret** of the `SetupIntent`.

### data optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

[× Hide data properties](#)

#### payment\_method RECOMMENDED object | string

The `id` of an existing PaymentMethod or an object of collected data. See use cases below for details.

## Returns

`stripe.confirmSepaDebitSetup` will return a `Promise` which resolves with a `result` object.

This object has either:

- `result.setupIntent`: the successful `SetupIntent`.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmSepaDebitSetup` may take several seconds to complete.

During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm SEPA Debit setup with payment data from an Element

Create and attach a new `PaymentMethod` with `stripe.confirmSepaDebitSetup` by passing an `iban` `Element` to `payment_method[sepa_debit]`. The new `PaymentMethod` will be created with the data collected by the `Element` and will be used to confirm the `SetupIntent`. Additionally, to create a SEPA Direct Debit `PaymentMethod`, you are required to collect and include the customer's name and email address.

### Data argument properties

#### `payment_method` REQUIRED object

Pass an `object` to confirm using data collected by an `iban` `Element`.

##### `× Hide payment_method properties`

###### `sepa_debit` REQUIRED Element

An `iban` `Element`.

###### `billing_details` REQUIRED Object

The customer's `billing_details` . `name` and `email` are required.

###### `× Hide billing_details properties`

###### `name` REQUIRED string

The customer's name.

###### `email` REQUIRED string

The customer's email.

## Confirm SEPA Debit setup with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmSepaDebitSetup` and it will be used to confirm the `SetupIntent`.

### Data argument properties

---

#### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

## Confirm SEPA Debit setup with self collected data

If you already know the customer's IBAN account number or want to collect it yourself, then you do not need to use the `iban` `Element`. You can pass in the customer's account number directly to create a new `PaymentMethod` and confirm the `SetupIntent`. Additionally, to create a SEPA Direct Debit `PaymentMethod`, you are required to collect and include the customer's name and email address.

### Data argument properties

---

#### `payment_method` REQUIRED object

Pass an `object` to confirm using data collected by an `iban` Element.

**x Hide payment\_method properties**

**sepa\_debit REQUIRED object**

An object of self-collected IBAN data.

**x Hide sepa\_debit properties**

**iban REQUIRED string**

An IBAN account number.

**billing\_details REQUIRED object**

The customer's **billing\_details** . `name` and `email` are required.

**x Hide billing\_details properties**

**name REQUIRED string**

The customer's name.

**email REQUIRED string**

The customer's email.

## `stripe.confirmSofortSetup(clientSecret, data?, options?)`

Use `stripe.confirmSofortSetup` in the [Set up future payments](#) flow to use SOFORT bank details to set up a SEPA Direct Debit payment method for future payments. When called, it will confirm a `SetupIntent` with `data` you provide, and it will automatically redirect the customer to authorize the transaction. Once authorization is complete, the customer will be redirected

back to your specified `return_url`. Note that there are some additional requirements to this flow that are not covered in this reference. Refer to our [integration guide](#) for more details.

When you confirm a `SetupIntent`, it needs to have an attached `PaymentMethod`. In addition to confirming the `SetupIntent`, this method can automatically create and attach a new `PaymentMethod` for you. It can also be called with an existing `PaymentMethod`, or if you have already attached a `PaymentMethod` you can call this method without needing to provide any additional data. These use cases are detailed in the sections that follow.

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `SetupIntent`.

### `data` optional object

Data to be sent with the request. Refer to the [Setup Intents API](#) for a full list of parameters.

#### `x Hide data properties`

##### `payment_method` RECOMMENDED object | string

The `id` of an existing `PaymentMethod` or an object of collected data. See use cases below for details.

##### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

### `options` optional object

An options object to control the behavior of this method.

[× Hide options properties](#)**handleActions** boolean

Set this to `false` if you want to [manually handle the authorization redirect](#). Default is `true`.

## Returns

`stripe.confirmSofortSetup` will return a `Promise` which resolves with a `result` object. This object has either:

- `result.setupIntent`: the successful [SetupIntent](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmSofortSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Confirm Sofort setup with an existing payment method

If you have already created a `PaymentMethod`, you can pass its `id` to `payment_method` when calling `stripe.confirmSofortSetup` and it will be used to confirm the `SetupIntent`.

## Data argument properties

---

### `payment_method` REQUIRED string

The `id` of an existing `PaymentMethod`.

---

### `return_url` RECOMMENDED string

The url your customer will be directed to after they complete authentication.

---

## Confirm Sofort setup with self collected data

Your customer's name, email and the country of their bank are required for the SOFORT authorization to succeed. You can pass in these properties directly to create a new `PaymentMethod` and confirm the `SetupIntent`.

## Data argument properties

---

### `payment_method` REQUIRED object

Pass an object to confirm with the customer's name and email.

### **x Hide payment\_method properties**

#### **country REQUIRED string**

The country code where customer's bank is located.

#### **billing\_details REQUIRED object**

The customer's **billing\_details** . `name` and `email` are required.

##### **x Hide billing\_details properties**

###### **name REQUIRED string**

The customer's name.

###### **email REQUIRED string**

The customer's email.

#### **return\_url RECOMMENDED string**

The url your customer will be directed to after they complete authentication.

## **stripe.retrieveSetupIntent(clientSecret)**

Retrieve a **SetupIntent** using its client secret.

### **Method parameters**

#### **clientSecret REQUIRED string**

The `client secret` of the `SetupIntent` to retrieve.

## Returns

---

This method returns a `Promise` which resolves with a `result` object. This object has either:

- `result.setupIntent` : a `SetupIntent` was retrieved successfully.
- `result.error` : an error. Refer to the [API reference](#) for all possible errors.

---

## `stripe.verifyMicrodepositsForSetup(clientSecret, data?)`

Use `stripe.verifyMicrodepositsForSetup` in the [Save details for future payments with pre-authorized debit in Canada](#) or [Save details for future payments with ACH Direct Debit](#) flow to verify a customer's bank account with micro-deposits.

It should be only called when `SetupIntent` is in the `requires_action` state, and contains a `next_action` field that has a `type` equal to `verify_with_microdeposits`. Refer to our [integration guide](#) for more details.

---

## Method parameters

### `clientSecret` REQUIRED string

The `client secret` of the `SetupIntent`.

---

### `data` optional object

Data to be sent with the request.

[× Hide data properties](#)**amounts** array

An array of two positive integers, in *cents*, equal to the values of the micro-deposits sent to the bank account.

**descriptor\_code** string

A six-character code starting with SM present in the microdeposit sent to the bank account.

## Returns

`stripe.verifyMicrodepositsForSetup` will return a `Promise` which resolves with a result object. This object has either:

- `result.setupIntent` : the [SetupIntent](#) with a `status` of `succeeded`.
- `result.error` : an error. Refer to the [API reference](#) and our [integration guide](#) for all possible errors.

**i** Verification can fail for several reasons. The failure may happen synchronously as a direct error response, or asynchronously through a `payment_intent.payment_failed` webhook event. Refer to our [integration guide](#) for more details.

## [`stripe.handleNextAction\(options\)`](#)

Use `stripe.handleNextAction` in the [finalizing payments on the server](#) flow to finish confirmation of a [`SetupIntent`](#) with the `requires_action` status. It will throw an error if the `SetupIntent` has a different status.

Depending on the payment method and required action, the customer may be temporarily redirected from your site and brought back to the `return_url` [parameter](#) provided when the `SetupIntent` is confirmed.

## Method parameters

`options` REQUIRED object

× Hide options properties

`clientSecret` string

The [client secret](#) of the `SetupIntent`.

## Returns

This method returns a `Promise` which resolves with a `result` object. This object has either:

- `result.setupIntent`: a [`SetupIntent`](#) with the `processing` or `succeeded` status.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

ⓘ Note that `stripe.handleNextAction` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.handleNextAction` may trigger a **3D Secure** authentication challenge. The authentication challenge requires a context switch that can be hard to follow on a screen-reader. Ensure that your form is accessible by ensuring that success or error messages are clearly read out.

## `stripe.collectBankAccountForSetup(options)`

Use `stripe.collectBankAccountForSetup` in the **Save bank details** flow for the **ACH Direct Debit** payment method to collect the customer's bank account in your payment form. When called, it will automatically load an on-page modal UI to collect bank account details and verification, and attach the **PaymentMethod** to the **SetupIntent**.

### Method parameters

#### `options` REQUIRED object

Data to be sent with the request.

[× Hide options properties](#)

**clientSecret** REQUIRED string

The **client secret** of the `SetupIntent`.

**params** REQUIRED object

[× Hide params properties](#)

**payment\_method\_type** REQUIRED string

The payment method type for the bank account details (e.g. `us_bank_account`)

**payment\_method\_data** REQUIRED object

Payment method specific data to be sent with the request

[+ Show payment\\_method\\_data properties](#)

## Returns

When the `stripe.collectBankAccountForSetup` completes successfully, it returns a `SetupIntent`. If the customer provided their account, the `SetupIntent` is in the `requires_confirmation` state. If the customer closed the dialog without providing their account, the `SetupIntent` is in the `requires_payment_method` state. Use `stripe.confirmUsBankAccountSetup` to complete the process.

# The Payment Request object

A `PaymentRequest` object is used to collect payment information through an interface controlled and styled by the browser itself (i.e., not by you or your page).

See the [Payment Request Button Element quickstart](#) for a high-level overview of when you'd want to do this.

---

## `stripe.paymentRequest(options)`

Use `stripe.paymentRequest` to create a `PaymentRequest` object. Creating a `PaymentRequest` requires that you configure it with an `options` object.

In Safari, `stripe.paymentRequest` uses Apple Pay, and in other browsers it uses the [Payment Request API standard](#).

### Method parameters

---

#### `options` REQUIRED object

A set of options to create this `PaymentRequest` instance with. These options can be updated using `paymentRequest.update`.

[X Hide options properties](#)**country** REQUIRED string

The two-letter country code of your Stripe account (e.g., `US`).

**currency** REQUIRED string

Three character currency code (e.g., `usd`).

**total** REQUIRED PaymentItem

A [PaymentItem](#) object. This `PaymentItem` is shown to the customer in the browser's payment interface.

**displayItems** array

An array of [PaymentItem](#) objects. These objects are shown as line items in the browser's payment interface. Note that the sum of the line item amounts does not need to add up to the `total` amount above.

**requestPayerName** RECOMMENDED boolean

By default, the browser's payment interface only asks the customer for actual payment information. A customer name can be collected by setting this option to `true`. This collected name will appear in the [PaymentResponse](#) object.

We highly recommend you collect name as this also results in collection of billing address for Apple Pay. The billing address can be used to perform address verification and block fraudulent payments. For all other payment methods, the billing address is automatically collected when available.

**requestPayerEmail** boolean

See the `requestPayerName` option.

**requestPayerPhone** boolean

See the `requestPayerName` option.

**requestShipping** boolean

Collect shipping address by setting this option to `true`. The address appears in the [PaymentResponse](#).

You must also supply a valid [ShippingOptions] to the `shippingOptions` property. This can be up front at the time `stripe.paymentRequest` is called, or in response to a `shippingaddresschange` event using the `updateWith` callback.

**shippingOptions** array

An array of [ShippingOption](#) objects. The first shipping option listed appears in the browser payment interface as the default option.

**disableWallets** array

An array of wallet strings. Can be one or more of `applePay`, `googlePay`, `link`, and `browserCard`. Use this option to disable Apple Pay, Google Pay, Link, and/or browser-saved cards.

 `stripe.paymentRequest` abstracts over a number of implementation details to work uniformly across Apple Pay and the Payment Request browser standard. In particular, under the hood we set `supportedNetworks` to its most permissive setting, dynamically accounting for country and platform. It is currently not possible to override this and make `supportedNetworks` be more restrictive.

## **paymentRequest.canMakePayment()**

Returns a `Promise` that resolves with an object detailing if an enabled wallet is ready to pay. If no wallet is available, it resolves with `null`. The resolution object has the properties in the table

below.

**NOTE:** The `paymentRequestButton` element automatically shows the correct wallet branding. You shouldn't need to inspect the return object's properties unless you are building your own custom button.

## Return object properties

---

### `applePay` optional boolean

`true` if Apple Pay wallet is ready to pay. In this case:

- `paymentRequestButton` Element will show as a branded Apple Pay button automatically.
- When using a custom button, you'll want to show a button that conforms to the Apple Pay [Human Interface Guidelines](#).

---

### `googlePay` optional boolean

`true` if Google Pay wallet is ready to pay. In this case:

- `paymentRequestButton` Element will show as a branded Google Pay button automatically.
- When using a custom button, you'll want to show a button that conforms to the Google Pay [Brand Guidelines](#).

---

### `link` optional boolean

`true` if Link wallet is ready to pay. In this case:

- `paymentRequestButton` Element will show as a branded Link button automatically.
- Link is not supported in custom button configurations.



`canMakePayment` resolves to `null` outside the following supported cases:

- Safari 10.1+ (desktop and mobile)
  - with a saved Apple Pay card

- or when in a Private Browsing window
- or when the “Allow websites to check if Apple Pay is set up” preference is disabled
- Chrome 61+ (desktop and mobile)
  - with a saved Google Pay card
  - or when the browser has a saved card (i.e. autofill)

For more information, see [Testing your integration](#).

## `paymentRequest.show()`

Shows the browser’s payment interface. When using the `paymentRequestButton` Element, this is called for you automatically. This method must be called as the result of a user interaction (for example, in a click handler).

## `paymentRequest.update(options)`

`PaymentRequest` instances can be updated with an options object. Available options are documented below.

`paymentRequest.update` can only be called when the browser payment interface is not showing. Listen to the `click` and `cancel` events to detect if the payment interface has been initiated. To update the `PaymentRequest` right before the payment interface is initiated, call `paymentRequest.update` in your click event handler.

## Method parameters

**options** REQUIRED object

A set of options to update this `PaymentRequest` instance with.

[X Hide options properties](#)

**currency** string

Three character currency code (e.g., `usd`).

**total** object

A `PaymentItem` object. This `PaymentItem` is shown to the customer in the browser's payment interface.

**displayItems** array

An array of `PaymentItem` objects. These payment items are shown as line items in the browser's payment interface. Note that the sum of the line item amounts does not need to add up to the `total` amount above.

**shippingOptions** array

An array of `ShippingOption` objects. The first shipping option listed appears in the browser payment interface as the default option.

## PaymentRequest events

`PaymentRequest` instances emit several different types of events.

## paymentRequest.on('token', handler)

Stripe.js automatically creates a [Token](#) after the customer is done interacting with the browser's payment interface. To access the created [Token](#), listen for this event.

### Method parameters

---

#### event REQUIRED string

The name of the event. In this case, `token`.

---

#### handler REQUIRED function

A callback function that will be called with a [PaymentResponse](#) object when the event is fired.

The [PaymentResponse](#) object will contain a `token` field.

---

## paymentRequest.on('paymentmethod', handler)

Stripe.js automatically creates a [PaymentMethod](#) after the customer is done interacting with the browser's payment interface. To access the created [PaymentMethod](#), listen for this event.

### Method parameters

---

#### event REQUIRED string

The name of the event. In this case, `paymentmethod`.

---

#### handler REQUIRED function

A callback function that will be called with a [PaymentResponse](#) object when the event is fired.

The `PaymentResponse` object will contain a `paymentMethod` field.

---

## `paymentRequest.on('source', handler)`

Stripe.js automatically creates a [Source](#) after the customer is done interacting with the browser's payment interface. To access the created source, listen for this event.

### Method parameters

---

**event** REQUIRED string

The name of the event. In this case, `source`.

---

**handler** REQUIRED function

A callback function that will be called with a [PaymentResponse](#) object when the event is fired.

The `PaymentResponse` object will contain a `source` field.

---

## `paymentRequest.on('cancel', handler)`

The `cancel` event is emitted from a [PaymentRequest](#) when the browser's payment interface is dismissed.

Note that in some browsers, the payment interface may be dismissed by the customer even after they authorize the payment. This means that you may receive a `cancel` event on your `PaymentRequest` object after receiving a `token`, `paymentmethod`, or `source` event. If you're

using the `cancel` event as a hook for canceling the customer's order, make sure you also refund the payment that you just created.

## Method parameters

---

### **event** REQUIRED string

The name of the event. In this case, `cancel`.

---

### **handler** REQUIRED function

A callback function that you will provide that will be called when the event is fired.

---

## `paymentRequest.on('shippingaddresschange', handler)`

The `shippingaddresschange` event is emitted from a `PaymentRequest` whenever the customer selects a new address in the browser's payment interface.

## Method parameters

---

### **event** REQUIRED string

The name of the event. In this case, `shippingaddresschange`.

---

### **handler** REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired.

When called it will be passed an event object with the following properties:

## [X Hide handler event object properties](#)

### **updateWith** function

`updateWith(updateDetails) => void` is a Stripe.js provided function that is called with an **UpdateDetails** object to merge your updates into the current **PaymentRequest** object. Note that if you subscribe to `shippingaddresschange` events, then you must call `updateWith` within 30 seconds.

### **shippingAddress** `ShippingAddress`

The customer's selected **ShippingAddress**.

To maintain privacy, browsers may anonymize the shipping address by removing sensitive information that is not necessary to calculate shipping costs. Depending on the country, some fields can be missing or partially redacted. For example, the shipping address in the U.S. may only contain a city, state, and ZIP code. The full shipping address appears in the **PaymentResponse** object after the purchase is confirmed in the browser's payment interface

## `paymentRequest.on('shippingoptionchange', handler)`

The `shippingoptionchange` event is emitted from a **PaymentRequest** whenever the customer selects a new shipping option in the browser's payment interface.

### Method parameters

#### **event** REQUIRED string

The name of the event. In this case, `shippingoptionchange`.

#### **handler** REQUIRED function

`handler(event) => void` is a **callback function** that you provide that will be called when the event is fired.

When called it will be passed an event object with the following properties:

[X Hide handler event object properties](#)

#### **updateWith** function

`updateWith(updateDetails) => void` is a Stripe.js provided function that is called with an **UpdateDetails** object to merge your updates into the current **PaymentRequest** object. Note that if you subscribe to `shippingoptionchange` events, then you must call `updateWith` within 30 seconds.

#### **shippingOption** `ShippingOption`

The customer's selected **ShippingOption**.

## Payment Methods

PaymentMethod objects represent your customer's payment instruments. They can be used with PaymentIntents to collect payments or saved to Customer objects to store instrument details for future payments.

Related guides: [Payment Methods](#) and [More Payment Scenarios](#).

`stripe.createPaymentMethod(paymentMethodData)`

Use `stripe.createPaymentMethod` to convert payment information collected by elements into a [PaymentMethod](#) object that you safely pass to your server to use in an API call.

**NOTE:** In most integrations, you will not need to use this method. Instead, use methods like `stripe.confirmCardPayment`, which will automatically create a PaymentMethod when you confirm a [PaymentIntent](#).

## Method parameters

---

`paymentMethodData` **REQUIRED** object

Refer to the [PaymentMethod API](#) for a full list of parameters.

[X Hide paymentMethodData properties](#)

**type** REQUIRED string

The type of the [PaymentMethod](#) to create. Refer to the [PaymentMethod API](#) for all possible values.

**card** Element

A `card` or `cardNumber` Element.

**au\_becs\_debit** Element

An `auBankAccount` Element.

**fpx** Element

An `fpx` Element.

**fpx[bank]** string

The customer's [bank](#).

**netbanking[bank]** string

The customer's bank.

**ideal** Element

An `idealBank` Element.

**ideal[bank]** string

The customer's [bank](#).

**sepa\_debit** Element

An `iban` Element.

**sepa\_debit[iban]** string

An IBAN account number.

**upi[vpa]** string

The customer's VPA.

**billing\_details** object

**Billing information** associated with the PaymentMethod that may be used or required by particular types of payment methods.

## Returns

`stripe.createPaymentMethod(paymentMethodData)` returns a `Promise` which resolves with a result object. This object has either:

- `result.paymentMethod`: a **PaymentMethod** was created successfully.
- `result.error`: there was an error. This includes client-side validation errors. Refer to the [API reference](#) for all possible errors.

## `stripe.createPaymentMethod(options)`

Use `stripe.createPaymentMethod` to convert payment information collected by elements into a **PaymentMethod** object that you safely pass to your server to use in an API call.

**NOTE:** In most integrations, you will not need to use this method. Instead, use methods like `stripe.confirmPayment`, which will automatically create a PaymentMethod when you confirm a `PaymentIntent`.

## Method parameters

**options** REQUIRED object

[× Hide options properties](#)

**elements** REQUIRED object

The [Elements](#) instance that was used to create the Payment Element. It will be used to pull payment method and billing address data from.

**params** object

Parameters that will be passed on to the Stripe API. Refer to the [PaymentMethod API](#) for a full list of parameters.

[+ Show params properties](#)

## Returns

`stripe.createPaymentMethod(options)` returns a [Promise](#) which resolves with a result object.

This object has either:

- `result.paymentMethod`: a [PaymentMethod](#) was created successfully.
- `result.error`: there was an error. This includes client-side validation errors. Refer to the [API reference](#) for all possible errors.

## Tokens

Stripe uses tokens to collect sensitive client information without exposing it. When you use the [Payment Intents API](#), the client handles tokenization and you don't have to create tokens yourself.

Stripe.js provides the following method to create [Tokens](#) .

---

```
stripe.createToken(cardElement , data?)
```

Use `stripe.createToken` to convert information collected by card elements into a single-use [Token](#) that you safely pass to your server to use in an API call.

## Method parameters

---

### `cardElement` REQUIRED Element

The `card` [Element](#) you wish to tokenize data from. If applicable, the `Element` tokenizes by pulling data from other elements you've created on the same instance of [Elements](#)—you only need to supply one `Element` as the parameter.

---

### `data` optional object

An object containing additional payment information you might have collected.

Although these fields are optional, we highly recommend collecting name and address. This information can be used to perform a number of verifications, such as CVC, ZIP, and address verification. [Radar](#) includes [built-in](#) rules that can block payments where the ZIP or CVC verifications with the cardholder's bank failed.

[× Hide data properties](#)**name** RECOMMENDED string

Cardholder name.

**address\_line1** string**address\_line2** string**address\_city** string**address\_state** string**address\_zip** string**address\_country** RECOMMENDED stringA two character country code (for example, `us`).**currency** stringRequired in order to **add the card to a Connect account** (in all other cases, this parameter is not used).

## Returns

`stripe.createToken` returns a `Promise` which resolves with a `result` object. This object has either:

- `result.token`: a **Token** was created successfully.
- `result.error`: there was an error. This includes client-side validation errors. Refer to the [API reference](#) for all possible errors.

## stripe.verifyIdentity(clientSecret)

Use `stripe.verifyIdentity` to display an [Identity](#) modal that securely collects verification information.

### Method parameters

**clientSecret** REQUIRED string

The [client secret](#) of the [VerificationSession](#).

### Returns

This method returns a [Promise](#) which resolves with a result object. If this method fails, the result object will contain a localized error message in the `error.message` field.

## stripe.createEphemeralKeyNonce(options)

Use `stripe.createEphemeralKeyNonce` in the [Issuing Elements](#) flow to create an ephemeral key nonce. Refer to our [integration guide](#) for more details.

### Method parameters

**options** REQUIRED object

Data to be sent with the request.

[X Hide options properties](#)

**issuingCard** REQUIRED string

The `id` of an existing [Issuing card](#). The nonce returned by this method can only be used to retrieve this Issuing card.

## Returns

`stripe.createEphemeralKeyNonce` will return a `Promise` which resolves with a `result` object.

This object has either:

- `result.nonce`: the successful ephemeral key nonce.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

## `stripe.retrieveIssuingCard(issuingCard, options)` DEPRECATED

Use `stripe.retrieveIssuingCard` in the [Issuing Elements](#) flow to retrieve an Issuing card. Note that once you retrieve the card, you still need to display the card details in Elements. You need to pass in the `issuing_elements_2` beta to the Stripe instance to access this method. Refer to our [integration guide](#) for more details.

## Method parameters

**issuingCard** REQUIRED string

The `id` of an existing [Issuing card](#).

**options** REQUIRED object

An options object to configure the Issuing card retrieved from this method.

[X Hide options properties](#)

**ephemeralKeySecret** REQUIRED string

The ephemeral key secret retrieved from the API. The card `id` used to retrieve the ephemeral key must match the card `id` passed into this method. See the [integration guide](#) for more details.

**nonce** REQUIRED string

The ephemeral key nonce returned by `stripe.createEphemeralKeyNonce`. The card `id` used to create the nonce must match the card `id` passed into this method.

**expand** array

An array of Card fields to expand in the response from the Stripe API. Can be one or more of `number`, `cvc`, and `pin.number`. Expand fields as required to display Issuing Elements for those fields.

## Returns

`stripe.retrieveIssuingCard` will return a `Promise` which resolves with a `result` object. This object has either:

- `result.issuingCard`: the successful [Issuing card](#).
- `result.error`: an error. Refer to the [API reference](#) for all possible errors.

## Financial Connections

Stripe [Financial Connections](#) enables your users to securely share their financial data by linking their external financial accounts to your business.

### `stripe.collectFinancialConnectionsAccounts(options)`

Use `stripe.collectFinancialConnectionsAccounts` in the [Add a Financial Connections Account to retrieve data](#) flow. When called, it will load the [Authentication Flow](#), an on-page modal UI which allows your user to securely link their external financial account(s).

#### Method parameters

`options` REQUIRED object

× Hide options properties

`clientSecret` REQUIRED string

The `client_secret` of the [Financial Connections Session](#).

#### Returns

`stripe.collectFinancialConnectionsAccounts` returns a `Promise` which resolves with a `result` object. The object has either:

- `result.financialConnectionsSession`: the updated **Financial Connections Session** object. The `accounts` property will contain the list of accounts collected during this session. If the user chooses to link no accounts, or exits the Authentication Flow early, the `accounts` array will be empty.
- `result.error`: there was an error launching the Authentication Flow. For example, the Financial Connections Session has expired, or the Financial Connections Session has already been used.

## `stripe.collectBankAccountToken(options)`

Use `stripe.collectBankAccountToken` in the **Add a Financial Connections Account to a US Custom Connect account** flow. When called, it will load the **Authentication Flow**, an on-page modal UI which allows your user to securely link their external financial account for payouts.

### Method parameters

`options` **REQUIRED** object

[× Hide options properties](#)

`clientSecret` **REQUIRED** string

The `client_secret` of the **Financial Connections Session**.

### Returns

`stripe.collectBankAccountToken` returns a `Promise` which resolves with a `result` object.

If an external financial account was successfully collected, the result object will contain:

- `result.token`: a bank account **Token**.
- `result.financialConnectionsSession`: the updated **Financial Connections Session** object. The `accounts` array will contain a single Financial Connections Account (representing the same external financial account used for `result.token`).

If there was an error launching the Authentication Flow, the result object will contain:

- `result.error`: there was an error launching the Authentication Flow. For example, the Financial Connections Session has expired, or the Financial Connections Session has already been used.

If the user chooses to link no accounts, or exits the Authentication Flow early, the result object will contain:

- `result.financialConnectionsSession`: the updated **Financial Connections Session** object. The `accounts` array will be empty.

---

## The `CssFontSource` object

This object is used to pass custom fonts via a stylesheet URL when creating an **Elements** object.

### Object Properties

---

`cssSrc` string

A relative or absolute URL pointing to a CSS file with `@font-face` definitions, for example:

```
https://fonts.googleapis.com/css?family=Open+Sans
```

Note that if you are using a [content security policy](#) (CSP), [additional directives](#) may be necessary.

---

## The CustomFontSource object

This object is used to pass custom fonts when creating an [Elements](#) object.

### Object Properties

---

**family** REQUIRED string

The name to give the font.

---

**src** REQUIRED string

A valid **src** value pointing to your custom font file. This is usually (though not always) a link to a file with a `.woff`, `.otf`, or `.svg` suffix.

---

**display** string

A valid [font-display](#) value.

---

**style** string

One of `normal`, `italic`, `oblique`. Defaults to `normal`.

---

**unicodeRange** string

A valid **unicode-range** value.

---

**weight** string

A valid **font-weight**. Note that this is a string, not a number.

---

## The Contact object

The `Contact` object describes a saved contact passed in to the [Address Element](#).

### Object Properties

---

**name** string

The name of the contact. This might be a person, or a business name.

---

**address** object

The address of the contact.

[X Hide address properties](#)

**line1** REQUIRED string

**line2** string

**city** REQUIRED string

The name of a city, town, village, etc.

**state** REQUIRED string

The most coarse subdivision of a country. Depending on the country, this might correspond to a state, a province, an oblast, a prefecture, or something else along these lines.

**postal\_code** REQUIRED string

The postal code or ZIP code, also known as PIN code in India.

**country** REQUIRED string

Two-letter country code, capitalized. Valid two-letter country codes are specified by ISO3166 alpha-2.

**phone** string

The phone number of the contact.

The **fields.phone** option must be set to `always` if this property is specified.

## The PaymentItem object

The `PaymentItem` object is used to configure a [PaymentRequest](#).

## Object Properties

---

### `amount` REQUIRED number

The amount in the currency's subunit (e.g. cents, yen, etc.)

---

### `label` REQUIRED string

A name that the browser shows the customer in the payment interface.

---

### `pending` boolean

If you might change this amount later (for example, after you have calculated shipping costs), set this to `true`. Note that browsers treat this as a hint for how to display things, and not necessarily as something that will prevent submission.

---

## The PaymentResponse object

This object is returned as the payload of the `PaymentRequest` object's `token`, `paymentmethod`, or `source` event handlers.

## Object Properties

---

### `token` object

A [Token](#) object. Present if this was the result of a `token` event listener.

---

**paymentMethod object**

A [PaymentMethod](#) object. Present if this was the result of a `paymentmethod` event listener.

---

**source object**

A [Source](#) object. Present if this was the result of a `source` event listener.

---

**complete function**

`complete(status) => void` is a Stripe.js provided function. Call this when you have processed the token data provided by the API. Note that you must call `complete` within 30 seconds.

Accepts one of the following values:

[× Hide allowed status values](#)

**'success'** value

Report to the browser that the payment was successful, and that it can close any active payment interface.

**'fail'** value

Report to the browser that you were unable to process the customer's payment. Browsers may re-show the payment interface, or simply show a message and close.

**'invalid\_payer\_name'** value

Equivalent to `fail`, except that the browser can choose to show a more-specific error message.

**'invalid\_payer\_phone'** value

Equivalent to `fail`, except that the browser can choose to show a more-specific error message.

**'invalid\_payer\_email'** value

Equivalent to `fail`, except that the browser can choose to show a more-specific error message.

**'invalid\_shipping\_address'** value

Equivalent to `fail`, except that the browser can choose to show a more-specific error message.

**payerName** string

The customer's name. Only present if it was explicitly [asked for](#) when [creating the PaymentRequest object](#).

**payerEmail** string

The customer's email. Only present if it was explicitly **asked for** when **creating the PaymentRequest object**.

**payerPhone** string

The customer's phone. Only present if it was explicitly **asked for** when **creating the PaymentRequest object**.

**shippingAddress** ShippingAddress

The final **ShippingAddress** the customer selected.

Only populated when `requestShipping` is `true` when **creating the PaymentRequest object**, and you've supplied at least one `ShippingOption`.

**shippingOption** ShippingOption

The final **ShippingOption** the customer selected.

Only present when `requestShipping` is `true` when **creating the PaymentRequest object**, and you've supplied at least one `ShippingOption`.

**walletName** string

The unique name of the wallet the customer chose to authorize payment. For example, `browserCard`.

## The ShippingOption object

The `ShippingOption` object describes a shipping method used with a **PaymentRequest**.

## Object Properties

### **id** string

A unique ID you create to keep track of this shipping option. You'll be told the ID of the selected option on changes and on completion.

### **label** string

A short label for this shipping option.

### **detail** string

A longer description of this shipping option.

### **amount** number

The amount to show for this shipping option. If the cost of this shipping option depends on the shipping address the customer enters, listen for the `shippingaddresschange` event.

## The ShippingAddress object

The `ShippingAddress` object describes a shipping address collected with a [PaymentRequest](#).

## Object Properties

### **country** string

Two-letter country code, capitalized. Valid two-letter country codes are specified by ISO3166 alpha-2.

---

**addressLine** array

An array of address line items. For example, `185 Berry St.`, `Suite 500`, `P.O. Box 12345`, etc.

---

**region** string

The most coarse subdivision of a country. Depending on the country, this might correspond to a state, a province, an oblast, a prefecture, or something else along these lines.

---

**city** string

The name of a city, town, village, etc.

---

**postalCode** string

The postal code or ZIP code, also known as PIN code in India.

---

**recipient** string

The name of the recipient. This might be a person, a business name, or contain "care of" (c/o) instructions.

---

**phone** string

The phone number of the recipient. Note that this might be different from any phone number you collect with `requestPayerPhone`.

---

**sortingCode** string

The sorting code as used in, for example, France. Not present on Apple platforms.

---

**dependentLocality** string

A logical subdivision of a city. Can be used for things like neighborhoods, boroughs, districts, or UK dependent localities. Not present on Apple platforms.

---

## The Style object

Elements are styled using a `Style` object, which consists of CSS properties nested under objects for any of the following variants:

- `base`, base variant—all other variants inherit from these styles
- `complete`, applied when the Element has valid input
- `empty`, applied when the Element has no customer input
- `invalid`, applied when the Element has invalid input

The following pseudo-classes and pseudo-elements can also be styled using a nested object inside of a variant:

- `:hover`
- `:focus`
- `::placeholder`
- `::selection`
- `::-webkit-autofill`
- `::disabled`, available for all Elements except the `paymentRequestButton` Element.
- `::-ms-clear`, available for the `cardNumber`, `cardExpiry`, and `cardCvc` Elements. Inside the `::-ms-clear` selector, the `display` property can be customized.

The following CSS properties are supported:

## Supported CSS properties

---

### `backgroundColor` string

The `background-color` CSS property.

This property works best with the `::selection` pseudo-class. In other cases, consider setting the background color on the Element's container instead.

---

### `color` string

The `color` CSS property.

---

### `fontFamily` string

The `font-family` CSS property.

---

### `fontSize` string

The `font-size` CSS property.

---

### `fontSmoothing` string

The `font-smoothing` CSS property.

---

### `fontStyle` string

The `font-style` CSS property.

---

### `fontVariant` string

The `font-variant` CSS property.

---

### `fontWeight` string

The `font-weight` CSS property.

---

### `iconColor` string

A custom property, used to set the color of the icons that are rendered in an Element.

---

#### **lineHeight** string

The [line-height](#) CSS property.

To avoid cursors being rendered inconsistently across browsers, consider using a padding on the Element's container instead.

---

#### **letterSpacing** string

The [letter-spacing](#) CSS property.

---

#### **textAlign** string

The [text-align](#) CSS property.

Available for the `cardNumber`, `cardExpiry`, `cardCvc` and `iban` Elements.

---

#### **padding** string

The [padding](#) CSS property.

Available for the `idealBank` Element. Accepts integer length with `px` unit as values.

---

#### **textDecoration** string

The [text-decoration](#) CSS property.

---

#### **textShadow** string

The [text-shadow](#) CSS property.

---

#### **textTransform** string

The [text-transform](#) CSS property.

---

# The UpdateDetails object

This object is passed to the `updateWith` callback on a [PaymentRequest](#)'s `shippingaddresschange` and `shippingoptionchange` events.

## Object Properties

### `status` string

The browser uses this value to show an error message to the customer if they've taken an action that invalidates the payment request. The value must be one of the following:

[× Hide allowed values](#)

#### `'success'` value

Let the customer proceed.

#### `'fail'` value

Prevent the customer from making the change they just made.

#### `'invalid_shipping_address'` value

Equivalent to `fail`, except we show a more specific error message. Can only be used in a `shippingaddresschange` handler.

### `total` [PaymentItem](#)

The new total amount, if applicable, as a [PaymentItem](#) object.

### `displayItems` array

An array of [PaymentItem](#) objects. These PaymentItems are shown as line items in the browser's payment interface. Note that the sum of the line item amounts does not need to add up to the total amount above.

---

#### **shippingOptions** array

An array of [ShippingOption](#) objects. The first shipping option listed appears in the browser payment interface as the default option.

---

## Supported browsers

Stripe.js strives to support all recent versions of major browsers. For the sake of security and providing the best experience to the majority of customers, we do not support browsers that are no longer receiving security updates and represent a small minority of traffic. We support some older versions of evergreen browsers but recommend merchants keep their systems up to date for the best support. We support the following desktop browser versions:

- Chrome 38+
- Safari 10.1+
- Firefox 29+
- Edge 15+
- Opera 25+

We support the following mobile browsers:

- iOS Safari 9+ and other browsers and web views which use the system-provided WebKit engine
- Android Chrome 38+
- Samsung Browser 7.1+

For browsers not explicitly supported, we limit support as follows:

- We require TLS 1.2 to be supported by the browser.
- We require browsers that are sufficiently modern enough that they support [Promises](#) in the JavaScript programming language.
- We respond to bug reports but do not proactively test other browsers.

If you have an issue with Stripe.js on a specific browser, please [contact us](#) so we can improve its support.

---

## Supported locales

The following subset of [IETF language tags](#) can be used to configure localization in Stripe.js.

Note that Checkout supports a slightly different set of locales than Stripe.js. If you are using Checkout with `stripe.redirectToCheckout`, make sure to use a locale that Checkout supports.

VALUE	LOCALE	ELEMENTS	CHECKOUT
auto	Stripe detects the locale of the browser	✓	✓
ar	Arabic	✓	
bg	Bulgarian (Bulgaria)	✓	✓
cs	Czech (Czech Republic)	✓	✓
da	Danish (Denmark)	✓	✓
de	German (Germany)	✓	✓
el	Greek (Greece)	✓	✓
en	English	✓	✓
en-GB	English (United Kingdom)	✓	✓
es	Spanish (Spain)	✓	✓
es-419	Spanish (Latin America)	✓	✓

VALUE	LOCALE	ELEMENTS	CHECKOUT
et	Estonian (Estonia)	✓	✓
fi	Finnish (Finland)	✓	✓
fil	Filipino (Philippines)	✓	✓
fr	French (France)	✓	✓
fr-CA	French (Canada)	✓	✓
he	Hebrew (Israel)	✓	
hr	Croatian (Croatia)	✓	✓
hu	Hungarian (Hungary)	✓	✓
id	Indonesian (Indonesia)	✓	✓
it	Italian (Italy)	✓	✓
ja	Japanese (Japan)	✓	✓

VALUE	LOCALE	ELEMENTS	CHECKOUT
ko	Korean (Korea)	✓	✓
lt	Lithuanian (Lithuania)	✓	✓
lv	Latvian (Latvia)	✓	✓
ms	Malay (Malaysia)	✓	✓
mt	Maltese (Malta)	✓	✓
nb	Norwegian Bokmål	✓	✓
nl	Dutch (Netherlands)	✓	✓
pl	Polish (Poland)	✓	✓
pt-BR	Portuguese (Brazil)	✓	✓
pt	Portuguese (Brazil)	✓	✓
ro	Romanian (Romania)	✓	✓

VALUE	LOCALE	ELEMENTS	CHECKOUT
ru	Russian (Russia)	✓	✓
sk	Slovak (Slovakia)	✓	✓
sl	Slovenian (Slovenia)	✓	✓
sv	Swedish (Sweden)	✓	✓
th	Thai (Thailand)	✓	✓
tr	Turkish (Turkey)	✓	✓
vi	Vietnamese (Vietnam)	✓	✓
zh	Chinese Simplified (China)	✓	✓
zh-HK	Chinese Traditional (Hong Kong)	✓	✓
zh-TW	Chinese Traditional (Taiwan)	✓	✓

## Cookies

Stripe uses cookies to ensure that the site works properly, detect and prevent fraud, and understand how people interact with Stripe. There are some cookies that are essential for Stripe to function properly. These Necessary Cookies provide secure access to the site and enable page navigation. Other categories of cookies include Advertising Cookies, Analytics Cookies, and Preference Cookies.

You can find more information in the [Stripe Cookies Policy](#), including how to opt-out or manage your cookie preferences.

---

## Viewport meta tag requirements

In order to provide a great user experience for 3D Secure on all devices, you should set your page's viewport `width` to `device-width` with the [viewport meta tag](#).

There are several other viewport settings, and you can configure those based on your needs. Just make sure you include `width=device-width` somewhere in your configuration.

---

`stripe.redirectToCheckout(options?)` DEPRECATED

Use `stripe.redirectToCheckout` to redirect your customers to [Checkout](#), a Stripe-hosted page to securely collect payment information. When the customer completes their purchase, they are redirected back to your website.

## Method parameters

---

**options** optional object

[X Hide options properties](#)

**sessionId** string

The ID of the [Checkout Session](#) that is used in [Checkout's client and server integration](#).

**lineItems** array

An array of objects representing the items that your customer would like to purchase. These items are shown as line items in the Checkout interface and make up the total amount to be collected by Checkout. Used with the [client-only](#) integration.

[+ Show lineItems properties](#)

**mode** string

The mode of the Checkout Session, one of `payment` or `subscription`. Required if using `lineItems` with the [client-only](#) integration.

**successUrl** string

The URL to which Stripe should send customers when payment is complete. If you'd like access to the Checkout Session for the successful payment, read more about it in the guide on [fulfilling orders](#). Required if using the [client-only](#) integration.

**cancelUrl** string

The URL to which Stripe should send customers when payment is canceled. Required if using the [client-only](#) integration.

**clientReferenceId** string

A unique string to reference the Checkout session. This can be a customer ID, a cart ID, or similar. It is included in the `checkout.session.completed` webhook and can be used to fulfill the purchase.

**customerEmail** string

The email address used to create the customer object. If you already know your customer's

email address, use this attribute to prefill it on Checkout.

#### **billingAddressCollection** string

Specify whether Checkout should collect the customer's billing address. If set to `required`, Checkout will attempt to collect the customer's billing address. If not set or set to `auto` Checkout will only attempt to collect the billing address when necessary.

#### **shippingAddressCollection** object

When set, provides configuration for Checkout to collect a shipping address from a customer.

[+ Show shippingAddressCollection properties](#)

#### **locale** string

A **locale** that will be used to localize the display of Checkout. Default is `auto` (Stripe detects the locale of the browser).

#### **submitType** string

Describes the type of transaction being performed by Checkout in order to customize relevant text on the page, such as the **Submit** button. `submitType` can only be specified when using line items or SKUs, and not subscriptions. The default is `auto`. Supported values are: `auto` , `book` , `donate` , `pay` .

#### **items** array

An array of objects representing the items that your customer would like to purchase. These items are shown as line items in the Checkout interface and make up the total amount to be collected by Checkout. Using `lineItems` is preferred.

[+ Show items properties](#)

## Returns

This method returns a `Promise` which resolves with a result object. If this method fails, the result object will contain a localized error message in the `error.message` field.

- ⓘ If you are currently on a beta of the new version of Checkout, read the [Beta Migration Guide](#) to upgrade to the latest version.

## `stripe.handleCardPayment(clientSecret, element, data?)` DEPRECATED

`handleCardPayment` has been renamed to `confirmCardPayment`. In addition to the rename, we have slightly modified the arguments. These changes do not affect the behavior of the method. While we have no plans to ever remove support for `handleCardPayment`, we think the new name and arguments are easier to understand and better convey what the method is doing.

Use `stripe.handleCardPayment(clientSecret, element, data?)` when the customer submits your payment form. It will gather payment information from the element, which can be a `card` or `cardNumber` element, along with any other data you provide. It will then confirm the `PaymentIntent` and carry out 3DS or other `next_action`s if they are required.

If you are using `Dynamic 3D Secure`, `handleCardPayment` will trigger your Radar rules to execute and may open a dialog for your customer to authenticate their payment.

## Method parameters

`clientSecret` REQUIRED string

The **client secret** of the `PaymentIntent`.

---

**element** REQUIRED Element

A `card` or `cardNumber` **Element** that will be used to create a payment method.

---

**data** optional object

Data to be sent with the request.

[× Hide data properties](#)

**payment\_method\_data** object

Use this parameter to supply additional data relevant to the payment method, such as billing details.

[× Hide payment\\_method\\_data properties](#)

**billing\_details** RECOMMENDED object

The **billing details** associated with the card.

**shipping** RECOMMENDED object

The **shipping details** for the payment, if collected.

**receipt\_email** string

Email address that the receipt for the resulting payment will be sent to.

**setup\_future\_usage** string

Indicates that you intend to make future payments with this **PaymentIntent**'s payment method.

If present, the **PaymentMethod** used with this **PaymentIntent** can be **attached to a Customer**, even after the transaction completes.

Use **on\_session** if you intend to only reuse the **PaymentMethod** when your customer is present in your checkout flow. Use **off\_session** if your customer may or may not be in your checkout flow. See **saving card details during payment** to learn more.

Stripe uses **setup\_future\_usage** to dynamically optimize your payment flow and comply with regional legislation and network rules. For example, if your customer is impacted by SCA, using **off\_session** will ensure that they are authenticated while processing this **PaymentIntent**. You will then be able to collect **off-session payments** for this customer.

## Returns

A Promise which resolves with a `result` object.

The object will have either:

- `result.paymentIntent` : the successful [PaymentIntent](#).
- `result.error` : An error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.handleCardPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.handleCardPayment` may trigger a [3D Secure](#) authentication challenge. This will be shown in a modal dialog and may be confusing for customers using assistive technologies like screen readers. You should make your form accessible by ensuring that success or error messages are clearly read out after this method completes.

### `stripe.handleCardPayment(clientSecret, data?)` DEPRECATED

`handleCardPayment` has been renamed to [confirmCardPayment](#). In addition to the rename, we have slightly modified the arguments. These changes do not affect the behavior of the method. While we have no plans to ever remove support for `handleCardPayment`, we think the new name and arguments are easier to understand and better convey what the method is doing.

Use `stripe.handleCardPayment(clientSecret, data?)` to advance the [PaymentIntent](#) towards completion when you are not gathering payment method information from an [Element](#).

Call this variation when you have already attached a card to this [PaymentIntent](#) or if you want to attach an existing card to it.

## Method parameters

---

### **clientSecret** REQUIRED string

The [client secret](#) of the [PaymentIntent](#).

---

### **data** optional object

Data to be sent with the request.

[× Hide data properties](#)

**payment\_method** string

Use `payment_method` to specify an existing [PaymentMethod](#) to use for this payment.

Only one of `payment_method_data` and `payment_method` is required.

**payment\_method\_data** object

Use this parameter to supply additional data relevant to the payment method, such as billing details.

[× Hide payment\\_method\\_data properties](#)

**billing\_details** RECOMMENDED object

The [billing details](#) associated with the card.

**card[token]** string

Converts the provided token into a `PaymentMethod` to use for the payment.

**shipping** RECOMMENDED object

The [shipping details](#) for the payment, if collected.

**receipt\_email** string

Email address that the receipt for the resulting payment will be sent to.

**setup\_future\_usage** string

Indicates that you intend to make future payments with this [PaymentIntent](#)'s payment method.

If present, the `PaymentMethod` used with this [PaymentIntent](#) can be [attached to a Customer](#), even after the transaction completes.

Use `on_session` if you intend to only reuse the `PaymentMethod` when your customer is present in your checkout flow. Use `off_session` if your customer may or may not be in

your checkout flow. See [saving card details after a payment](#) to learn more.

Stripe uses `setup_future_usage` to dynamically optimize your payment flow and comply with regional legislation and network rules. For example, if your customer is impacted by SCA, using `off_session` will ensure that they are authenticated while processing this `PaymentIntent`. You will then be able to collect [off-session payments](#) for this customer.

## Returns

A Promise which resolves with a `result` object.

The object will have either:

- `result.paymentIntent`: the successful [PaymentIntent](#).
- `result.error`: An error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.handleCardPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.handleCardPayment` may trigger a [3D Secure](#) authentication challenge. This will be shown in a modal dialog and may be confusing for customers using assistive technologies like screen readers. You should make your form accessible by ensuring that success or error messages are clearly read out after this method completes.

```
stripe.confirmPaymentIntent(  
  clientSecret,  
  element,  
  data?  
) DEPRECATED
```

`confirmPaymentIntent` has been deprecated. Going forward, if you wish to confirm on the client without handling next actions, simply pass `{handleActions: false}` as a third argument to `confirmCardPayment`. While we have no plans to ever remove support for `confirmPaymentIntent`, we think that explicitly opting out of next action handling is easier to understand and will better convey what the method is doing.

Use `stripe.confirmPaymentIntent(clientSecret, element, data)` when the customer submits your payment form. It will gather payment information from element, along with any other data you provide, and confirm the `PaymentIntent`.

Only use this method if you want to **handle next actions yourself**. Otherwise, use `stripe.handleCardPayment`.

## Method parameters

---

### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent` to confirm.

---

### `element` REQUIRED Element

An `Element` that will be used to create a payment method.

---

### `data` optional object

Data to be sent with the request. It can contain the following parameters

[× Hide data properties](#)**return\_url** string

If you are [handling next actions yourself](#), pass in a `return_url`. If the subsequent action is `redirect_to_url`, this URL will be used on the return path for the redirect.

**payment\_method\_data** object

Use this parameter to supply additional data relevant to the payment method, such as billing details.

[× Hide payment\\_method\\_data properties](#)**billing\_details** RECOMMENDED object

The [billing details](#) associated with the card.

**shipping** RECOMMENDED object

The [shipping details](#) for the payment, if collected.

**receipt\_email** string

Email address that the receipt for the resulting payment will be sent to.

**setup\_future\_usage** string

Indicates that you intend to make future payments with this [PaymentIntent](#)'s payment method.

If present, the `PaymentMethod` used with this [PaymentIntent](#) can be [attached to a Customer](#), even after the transaction completes.

Use `on_session` if you intend to only reuse the `PaymentMethod` when your customer is present in your checkout flow. Use `off_session` if your customer may or may not be in your checkout flow. See [saving card details after a payment](#) to learn more.

Stripe uses `setup_future_usage` to dynamically optimize your payment flow and comply with regional legislation and network rules. For example, if your customer is impacted by

SCA, using `off_session` will ensure that they are authenticated while processing this `PaymentIntent`. You will then be able to collect **off-session payments** for this customer.

## Returns

A Promise which resolves with a `result` object.

The object will have either:

- `result.paymentIntent` : the successful **PaymentIntent** .
- `result.error` : An error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmPaymentIntent` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## `stripe.confirmPaymentIntent(clientSecret, data?)` DEPRECATED

`confirmPaymentIntent` has been deprecated. Going forward, if you wish to confirm on the client without handling next actions, simply pass `{handleActions: false}` as a third argument to `confirmCardPayment`. While we have no plans to ever remove support for `confirmPaymentIntent`, we think that explicitly opting out of next action handling is easier to understand and will better convey what the method is doing.

Use `stripe.confirmPaymentIntent(clientSecret, data)` to confirm the `PaymentIntent` when you are not gathering payment information from an `Element`. Call this variation when you have already attached a payment method to this `PaymentIntent`, or if you want to attach an existing card, token, or `PaymentMethod` to it.

Only use this method if you want to handle next actions yourself. Otherwise, use `stripe.handleCardPayment`.

## Method parameters

---

### **clientSecret** REQUIRED string

The `client secret` of the `PaymentIntent` to confirm.

---

### **data** optional object

Data to be sent with the request. It can contain the following parameters

### [× Hide data properties](#)

#### **return\_url** string

If you are [handling next actions yourself](#), pass in a `return_url`. If the subsequent action is `redirect_to_url`, this URL will be used on the return path for the redirect.

#### **payment\_method** string

Use `payment_method` to specify an existing [PaymentMethod](#) to use for this payment.

Only one of `payment_method_data` and `payment_method` is required.

#### **payment\_method\_data** object

Use this parameter to supply additional data relevant to the payment method, such as billing details.

##### [× Hide payment\\_method\\_data properties](#)

###### **billing\_details** RECOMMENDED object

The [billing details](#) associated with the card.

###### **card[token]** string

Converts the provided token into a `PaymentMethod` to use for the payment.

#### **shipping** RECOMMENDED object

The [shipping details](#) for the payment, if collected.

#### **receipt\_email** string

Email address that the receipt for the resulting payment will be sent to.

#### **setup\_future\_usage** string

Indicates that you intend to make future payments with this [PaymentIntent](#)'s payment method.

If present, the `PaymentMethod` used with this `PaymentIntent` can be **attached to a Customer**, even after the transaction completes.

Use `on_session` if you intend to only reuse the `PaymentMethod` when your customer is present in your checkout flow. Use `off_session` if your customer may or may not be in your checkout flow. See [saving card details after a payment](#) to learn more.

Stripe uses `setup_future_usage` to dynamically optimize your payment flow and comply with regional legislation and network rules. For example, if your customer is impacted by SCA, using `off_session` will ensure that they are authenticated while processing this `PaymentIntent`. You will then be able to collect **off-session payments** for this customer.

## Returns

A Promise which resolves with a `result` object.

The object will have either:

- `result.paymentIntent` : the successful [PaymentIntent](#).
- `result.error` : An error. Refer to the [API reference](#) for all possible errors.

 Note that `stripe.confirmPaymentIntent` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## `stripe.handleCardSetup(clientSecret, element, data?)` DEPRECATED

`handleCardSetup` has been renamed to `confirmCardSetup`. In addition to the rename, we have slightly modified the arguments. These changes do not affect the behavior of the method. While we have no plans to ever remove support for `handleCardSetup`, we think the new name and arguments are easier to understand and better convey what the method is doing.

Use `stripe.handleCardSetup(clientSecret, element, data?)` in the [Setup Intents API flow](#) when the customer submits your payment form. It will gather payment information from the `element`, which can be a `card` or `cardNumber` element, along with any other data you provide. It will then confirm the `SetupIntent` and carry out 3DS or other `next_action`s if they are required.

## Method parameters

---

### `clientSecret` REQUIRED string

The `client secret` of the `SetupIntent`.

---

### `element` REQUIRED Element

A `card` or `cardNumber` Element that will be used to create a payment method.

---

### `data` optional object

Data to be sent with the request.

× Hide data properties

**payment\_method\_data** object

Use this parameter to supply additional data relevant to the payment method, such as billing details.

× Hide payment\_method\_data properties

**billing\_details** RECOMMENDED object

The **billing details** associated with the card.

## Returns

A Promise which resolves with a `result` object.

The object will have either:

- `result.setupIntent`: the successful **SetupIntent**.
- `result.error`: An error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.handleCardSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.handleCardSetup` may trigger a **3D Secure** authentication challenge. This will be shown in a modal dialog and may be confusing for customers using assistive technologies like screen readers. You should make your form accessible

by ensuring that success or error messages are clearly read out after this method completes.

## stripe.handleCardSetup(`clientSecret`, `data?`) DEPRECATED

`handleCardSetup` has been renamed to `confirmCardSetup`. In addition to the rename, we have slightly modified the arguments. These changes do not affect the behavior of the method. While we have no plans to ever remove support for `handleCardSetup`, we think the new name and arguments are easier to understand and better convey what the method is doing.

Use `stripe.handleCardSetup(clientSecret, data)` to advance the `SetupIntent` towards completion when you are not gathering payment method information from an `Element`.

Call this variation when you have already attached a card to this `SetupIntent` or if you want to attach an existing card to it.

### Method parameters

#### `clientSecret` REQUIRED string

The `client secret` of the `SetupIntent`.

#### `data` optional object

Data to be sent with the request.

[× Hide data properties](#)

**payment\_method** string

Use `payment_method` to specify an existing [PaymentMethod](#) to use for this payment.

Only one of `payment_method_data` and `payment_method` is required.

**payment\_method\_data** object

Use this parameter to supply additional data relevant to the payment method, such as billing details.

[× Hide payment\\_method\\_data properties](#)

**billing\_details** RECOMMENDED object

The [billing details](#) associated with the card.

**card[token]** string

Converts the provided token into a `PaymentMethod` to use for the payment.

## Returns

A Promise which resolves with a `result` object.

The object will have either:

- `result.setupIntent` : the successful [SetupIntent](#).
- `result.error` : An error. Refer to the [API reference](#) for all possible errors.



Note that `stripe.handleCardSetup` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

Additionally, `stripe.handleCardSetup` may trigger a **3D Secure** authentication challenge. This will be shown in a modal dialog and may be confusing for customers using assistive technologies like screen readers. You should make your form accessible by ensuring that success or error messages are clearly read out after this method completes.

## `stripe.confirmSetupIntent(clientSecret, element, data?)` DEPRECATED

`confirmSetupIntent` has been deprecated. Going forward, if you wish to confirm on the client without handling next actions, simply pass `{handleActions: false}` as a third argument to `confirmCardSetup`. While we have no plans to ever remove support for `confirmSetupIntent`, we think that explicitly opting out of next action handling is easier to understand and will better convey what the method is doing.

Use `stripe.confirmSetupIntent(clientSecret, element, data)` when the customer submits your save payment method form. It will gather payment information from `element`, along with any other `data` you provide, and confirm the `SetupIntent`.

Only use this method if you want to **handle next actions yourself**. Otherwise, use `stripe.handleCardSetup`.

### Method parameters

`clientSecret` REQUIRED string

The **client secret** of the `SetupIntent` to confirm.

#### element REQUIRED Element

An **Element** that will be used to create a payment method.

#### data optional object

Data to be sent with the request. It can contain the following parameters

##### `x Hide data properties`

###### `payment_method_data` object

Use this parameter to supply additional data relevant to the payment method, such as billing details.

##### `x Hide payment_method_data properties`

###### `billing_details` RECOMMENDED object

The **billing details** associated with the card.

## Returns

A Promise which resolves with a `result` object.

The object will have either:

- `result.setupIntent`: the successful **SetupIntent**.
- `result.error`: An error. Refer to the **API reference** for all possible errors.



Note that `stripe.confirmSetupIntent` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## `stripe.confirmSetupIntent(clientSecret, data?)` DEPRECATED

`confirmSetupIntent` has been deprecated. Going forward, if you wish to confirm on the client without handling next actions, simply pass `{handleActions: false}` as a third argument to `confirmCardSetup`. While we have no plans to ever remove support for `confirmSetupIntent`, we think that explicitly opting out of next action handling is easier to understand and will better convey what the method is doing.

Use `stripe.confirmSetupIntent(clientSecret, data)` to confirm the `SetupIntent` when you are not gathering payment information from an `Element`. Call this variation when you have already attached a payment method to this `SetupIntent`, or if you want to attach an existing card, token, or `PaymentMethod` to it.

Only use this method if you want to handle next actions yourself. Otherwise, use `stripe.handleCardSetup`.

### Method parameters

#### `clientSecret` REQUIRED string

The `client secret` of the `SetupIntent` to confirm.

#### `data` optional object

Data to be sent with the request. It can contain the following parameters

[× Hide data properties](#)

`return_url` string

If you are [handling next actions yourself](#), pass in a `return_url`. If the subsequent action is `redirect_to_url`, this URL will be used on the return path for the redirect.

`payment_method` string

Use `payment_method` to specify an existing [PaymentMethod](#) to use for this payment.

Only one of `payment_method_data` and `payment_method` is required.

`payment_method_data` object

Use this parameter to supply additional data relevant to the payment method, such as billing details.

[× Hide payment\\_method\\_data properties](#)

`billing_details` RECOMMENDED object

The [billing details](#) associated with the card.

`card[token]` string

Converts the provided token into a `PaymentMethod` to use for the payment.

## Returns

A Promise which resolves with a `result` object.

The object will have either:

- `result.setupIntent` : the successful [SetupIntent](#).

- `result.error`: An error. Refer to the [API reference](#) for all possible errors.

**i** Note that `stripe.confirmSetupIntent` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## `stripe.handleFpxPayment(clientSecret, element, data?)` DEPRECATED

`handleFpxPayment` has been renamed to `confirmFpxPayment`. In addition to the rename, we have slightly modified the arguments. These changes do not affect the behavior of the method. While we will continue to support `handleFpxPayment` for the duration of the beta, we think the new name and arguments are easier to understand and better convey what the method is doing.

Use `stripe.handleFpxPayment` in the [FPX payment method creation](#) flow when the customer selects a bank from the dropdown. It will gather the `bank code` from the element, along with any other `PaymentIntent` `data` you provide. It will then create an FPX payment method and confirm the `PaymentIntent`.

### Method parameters

#### `clientSecret` REQUIRED string

The `client secret` of the `PaymentIntent`.

#### `element` REQUIRED Element

An `fpxBank` `Element` that will be used to create a payment method.

**data** optional object

A data object to be sent with the request.

[X Hide data properties](#)

**return\_url** string

The url your customer will be directed to after they complete authentication. Be sure to review the [payment confirmation page requirements](#).

## Returns

This method returns a `Promise` which resolves with a `result` object. This object has either:

- `result.paymentIntent`: a [PaymentIntent](#) with the `requires_confirmation` status to confirm server-side.
- `result.error`: an error. Refer to the [API reference](#) for all possible errors and the [FPX guide](#) for FPX specific errors.

 Note that `stripe.handleFpxPayment` may take several seconds to complete. During that time, you should disable your form from being resubmitted and show a waiting indicator like a spinner. If you receive an error result, you should be sure to show that error to the customer, re-enable the form, and hide the waiting indicator.

## Sources Deprecated

The Sources API has been deprecated and replaced by the [Payment Intents API](#).

Stripe.js provides the following methods to create and retrieve [Sources](#), which are part of the Charges API.

---

## stripe.createSource(`element`, `sourceData`) DEPRECATED

Use `stripe.createSource` to convert payment information collected by elements into a [Source](#) object that you safely pass to your server to use in an API call. See the [Sources documentation](#) for more information about sources.

### Method parameters

---

#### `element` REQUIRED object

The [Element](#) containing payment information. If applicable, the `Element` pulls data from other elements you've created on the same [Elements](#) instance.

---

#### `sourceData` REQUIRED object

A required object containing the `type` of [Source](#) you want to create, and any additional payment information that you have collected. See the [Sources API](#) reference for details.

---

### Returns

---

This method returns a `Promise` which resolves with a result object. This object has either:

- `result.source`: a [Source](#) was created successfully.

- `result.error`: there was an error. This includes client-side validation errors. Refer to the [API reference](#) for all possible errors.

## `stripe.createSource(sourceData)` DEPRECATED

Use `stripe.createSource` to convert raw payment information into a [Source](#) object that you can safely pass to your server for use in an API call. See the [Sources documentation](#) for more information about sources.

### Method parameters

#### `sourceData` REQUIRED object

A required object containing the `type` of `Source` you want to create, and any additional payment information that you have collected. See the [Sources API](#) reference for details.

### Returns

This method returns a `Promise` which resolves with a result object. This object has either:

- `result.source`: a [Source](#) was created successfully.
- `result.error`: there was an error. This includes client-side validation errors. Refer to the [API reference](#) for all possible errors.

 You cannot pass raw card information to `stripe.createSource(sourceData)`. Instead, you must gather card information in an Element and use

`stripe.createSource(element, sourceData)`. You can also pass an existing card token to convert it into a `Source` object.

## stripe.retrieveSource(source) DEPRECATED

Retrieve a `Source` using its unique ID and client secret.

### Method parameters

#### source REQUIRED object

An object containing the unique ID and client secret for a `Source`.

You can use a `Source` object created with `stripe.createSource` as the argument to `stripe.retrieveSource`, as every `Source` object has both `id` and `client_secret` keys.

#### × Hide source properties

##### `id` REQUIRED string

Unique identifier of the `Source`.

##### `client_secret` REQUIRED string

A secret available to the web client that created the `Source`, for purposes of retrieving the `Source` later from that same client.

### Returns

This method returns a `Promise` which resolves with a result object. This object has either:

- `result.source`: a **Source** was retrieved successfully.
- `result.error`: there was an error. This includes client-side validation errors. Refer to the [API reference](#) for all possible errors.















