Peter Gustafson
writing sample
Stripe Sep 18, 2013

Update a card | Stripe API Reference
Updates a specified card for a given customer.

Parameters

*   City/District/Suburb/Town/Village.

*   Billing address country, if provided when creating card.

*   Address line 1 (Street address/PO Box/Company name).

*   Address line 2 (Apartment/Suite/Unit/Building).

*   State/County/Province/Region.

*   ZIP or postal code.

*   Two digit number representing the card’s expiration month.

*   Four digit number representing the card’s expiration year.

*   Set of [key-value pairs](https://docs.stripe.com/api/metadata)
that you can attach to an object. This can be useful for storing
additional information about the object in a structured format.
Individual keys can be unset by posting an empty value to them. All
keys can be unset by posting an empty value to `metadata`.

*   Cardholder name.

POST /v1/customers/:id/sources/:id

```

curl https://api.stripe.com/v1/customers/acct_1032D82eZvKYlo2C/
sources/card_1NBLeN2eZvKYlo2CIq1o7Pzs \
  -u "sk_test_zzPhAh8...I4JDtzTNnhGlsk_test_zzPhAh8sZkhmI4JDtzTNnhGl:"
\
  -d name="Jenny Rosen"
```


```

{
  "id": "card_1NBLeN2eZvKYlo2CIq1o7Pzs",
```

```
  "object": "card",
  "address_city": null,
  "address_country": null,
  "address_line1": null,
  "address_line1_check": null,
  "address_line2": null,
  "address_state": null,
  "address_zip": null,
  "address_zip_check": null,
  "brand": "Visa",
  "country": "US",
  "cvc_check": "pass",
  "dynamic_last4": null,
  "exp_month": 8,
  "exp_year": 2024,
  "fingerprint": "Xt5EWLLDS7FJjR1c",
  "funding": "credit",
  "last4": "4242",
  "metadata": {},
  "name": "Jenny Rosen",
  "redaction": null,
  "tokenization_method": null,
  "wallet": null,
  "account": "acct_1032D82eZvKYlo2C"
}
```

You can always see the 10 most recent cards directly on a customer; this method lets you retrieve details about a specific card stored on the customer.

Parameters

No parameters.

Returns

Returns the `Card` object.

GET /v1/customers/:id/cards/:id

```

curl https://api.stripe.com/v1/customers/cus_NhD8HD2bY8dP3V/cards/
card_1MvoiELkdIwHu7ixOeFGbN9D \
  -u "sk_test_zzPhAh8...I4JDtzTNnhGlsk_test_zzPhAh8sZkhmI4JDtzTNnhGl:"
```

```


{
  "id": "card_1MvoiELkdIwHu7ixOeFGbN9D",
  "object": "card",
  "address_city": null,
  "address_country": null,
  "address_line1": null,
  "address_line1_check": null,
  "address_line2": null,
  "address_state": null,
  "address_zip": null,
  "address_zip_check": null,
  "brand": "Visa",
  "country": "US",
  "customer": "cus_NhD8HD2bY8dP3V",
  "cvc_check": null,
  "dynamic_last4": null,
  "exp_month": 4,
  "exp_year": 2024,
  "fingerprint": "mToisGZ01V71BCos",
  "funding": "credit",
  "last4": "4242",
  "metadata": {},
  "name": null,
  "tokenization_method": null,
  "wallet": null
}
```


You can see a list of the cards belonging to a customer. Note that the 10 most recent sources are always available on the `Customer` object. If you need more than those 10, you can use this API method and the `limit` and `starting_after` parameters to page through additional cards.

Parameters

No parameters.

More parameters

Returns

Returns a list of the cards stored on the customer.

GET /v1/customers/:id/cards

```
```

```
curl -G https://api.stripe.com/v1/customers/cus_NhD8HD2bY8dP3V/cards \
  -u "sk_test_zzPhAh8...I4JDtzTNnhGlsk_test_zzPhAh8sZkhmI4JDtzTNnhGl:"
\
  -d limit=3
```

```

{
  "object": "list",
  "url": "/v1/customers/cus_NhD8HD2bY8dP3V/cards",
  "has_more": false,
  "data": [
    {
      "id": "card_1MvoiELkdIwHu7ixOeFGbN9D",
      "object": "card",
      "address_city": null,
      "address_country": null,
      "address_line1": null,
      "address_line1_check": null,
      "address_line2": null,
      "address_state": null,
      "address_zip": null,
      "address_zip_check": null,
      "brand": "Visa",
      "country": "US",
      "customer": "cus_NhD8HD2bY8dP3V",
      "cvc_check": null,
      "dynamic_last4": null,
      "exp_month": 4,
      "exp_year": 2024,
      "fingerprint": "mToisGZ01V71BCos",
      "funding": "credit",
      "last4": "4242",
      "metadata": {},
      "name": null,
      "tokenization_method": null,
      "wallet": null
    }
    {...}
    {...}
  ],
}
```

You can delete cards from a customer. If you delete a card that is currently the default source, then the most recently added source will

become the new default. If you delete a card that is the last remaining source on the customer, then the default_source attribute will become null.

For recipients: if you delete the default card, then the most recently added card will become the new default. If you delete the last remaining card on a recipient, then the default_card attribute will become null.

Note that for cards belonging to customers, you might want to prevent customers on paid subscriptions from deleting all cards on file, so that there is at least one default card for the next invoice payment attempt.

Parameters

No parameters.

DELETE /v1/customers/:id/sources/:id

```

curl -X DELETE https://api.stripe.com/v1/customers/
acct_1032D82eZvKYlo2C/sources/card_1NGTaT2eZvKYlo2CZWSctn5n \
  -u "sk_test_zzPhAh8...I4JDtzTNnhGlsk_test_zzPhAh8sZkhmI4JDtzTNnhGl:"
```


```

{
  "id": "card_1NGTaT2eZvKYlo2CZWSctn5n",
  "object": "card",
  "deleted": true
}
```

`Source` objects allow you to accept a variety of payment methods. They represent a customer’s payment instrument, and can be used with the Stripe API just like a `Card` object: once chargeable, they can be charged, or can be attached to customers.

Stripe doesn’t recommend using the deprecated [Sources API](https://docs.stripe.com/api/sources). We recommend that you adopt the [PaymentMethods API](https://docs.stripe.com/api/payment_methods). This newer API provides access to our latest features and payment method types.

Related guides: [Sources API](https://docs.stripe.com/sources) and

[Sources & Customers](https://docs.stripe.com/sources/customers).

Prices define the unit cost, currency, and (optional) billing cycle for both recurring and one-time purchases of products. [Products](#products) help you track inventory or provisioning, and prices help you track payment terms. Different physical goods or levels of service should be represented by products, and pricing options should be represented by prices. This approach lets you change prices without having to change your provisioning scheme.

For example, you might have a single “gold” product that has prices for $10/month, $100/year, and €9 once.

Related guides: [Set up a subscription](https://docs.stripe.com/billing/subscriptions/set-up-subscription), [create an invoice](https://docs.stripe.com/billing/invoices/create), and more about [products and prices](https://docs.stripe.com/products-prices/overview).

A Promotion Code represents a customer-redeemable code for a [coupon](#coupons). It can be used to create multiple codes for a single coupon.

[Tax codes](https://stripe.com/docs/tax/tax-categories) classify goods and services for tax purposes.

Shipping rates describe the price of shipping presented to your customers and applied to a purchase. For more information, see [Charge for shipping](https://docs.stripe.com/payments/during-payment/charge-shipping).

A Checkout Session represents your customer’s session as they pay for one-time purchases or subscriptions through [Checkout](https://docs.stripe.com/payments/checkout) or [Payment Links](https://docs.stripe.com/payments/payment-links). We recommend creating a new Session each time your customer attempts to pay.

Once payment is successful, the Checkout Session will contain a reference to the [Customer](https://docs.stripe.com/api/customers), and either the successful [PaymentIntent](https://docs.stripe.com/api/payment_intents) or an active [Subscription](https://docs.stripe.com/api/subscriptions).

You can create a Checkout Session on your server and redirect to its URL to begin Checkout.

Related guide: [Checkout quickstart](https://docs.stripe.com/checkout/quickstart)

A payment link is a shareable URL that will take your customers to a

hosted payment page. A payment link can be shared and used multiple times.

When a customer opens a payment link it will open a new [checkout session](https://docs.stripe.com/api/checkout/sessions) to render the payment page. You can use [checkout session events](https://docs.stripe.com/api/events/types#event_types-checkout.session.completed) to track payments through payment links.

Related guide: [Payment Links API](https://docs.stripe.com/payment-links)

Issue a credit note to adjust an invoice’s amount after the invoice is finalized.

Related guide: [Credit notes](https://docs.stripe.com/billing/invoices/credit-notes)

Each customer has a [Balance](https://docs.stripe.com/api/customers/object#customer_object-balance) value, which denotes a debit or credit that’s automatically applied to their next invoice upon finalization. You may modify the value directly by using the [update customer API](https://docs.stripe.com/api/customers/update), or by creating a Customer Balance Transaction, which increments or decrements the customer’s `balance` by the specified `amount`.

Related guide: [Customer balance](https://docs.stripe.com/billing/customer/balance)

The Billing customer portal is a Stripe-hosted UI for subscription and billing management.

A portal configuration describes the functionality and features that you want to provide to your customers through the portal.

A portal session describes the instantiation of the customer portal for a particular customer. By visiting the session’s URL, the customer can manage their subscriptions and billing details. For security reasons, sessions are short-lived and will expire if the customer does not visit the URL. Create sessions on-demand when customers intend to manage their subscriptions and billing details.

Learn more in the [integration guide](https://docs.stripe.com/billing/subscriptions/integrating-customer-portal).

A portal configuration describes the functionality and behavior of a portal session.

Invoices are statements of amounts owed by a customer, and are either generated one-off, or generated periodically from a subscription.

They contain [invoice items](#invoiceitems), and proration adjustments that may be caused by subscription upgrades/downgrades (if necessary).

If your invoice is configured to be billed through automatic charges, Stripe automatically finalizes your invoice and attempts payment. Note that finalizing the invoice, [when automatic](https://docs.stripe.com/invoicing/integration/automatic-advancement-collection), does not happen immediately as the invoice is created. Stripe waits until one hour after the last webhook was successfully sent (or the last webhook timed out after failing). If you (and the platforms you may have connected to) have no webhooks configured, Stripe waits one hour after creation to finalize the invoice.

If your invoice is configured to be billed by sending an email, then based on your [email settings](https://dashboard.stripe.com/account/billing/automatic), Stripe will email the invoice to your customer and await payment. These emails can contain a link to a hosted page to pay the invoice.

Stripe applies any customer credit on the account before determining the amount due for the invoice (i.e., the amount that will be actually charged). If the amount due for the invoice is less than Stripe's [minimum allowed charge per currency](https://docs.stripe.com/currencies#minimum-and-maximum-charge-amounts), the invoice is automatically marked paid, and we add the amount due to the customer's credit balance which is applied to the next invoice.

More details on the customer's credit balance are [here](https://docs.stripe.com/billing/customer/balance).

Related guide: [Send invoices to customers](https://docs.stripe.com/billing/invoices/sending)

Invoice Items represent the component lines of an [invoice](https://docs.stripe.com/api/invoices). An invoice item is added to an invoice by creating or updating it with an `invoice` field, at which point it will be included as [an invoice line item](https://docs.stripe.com/api/invoices/line_item) within [invoice.lines](https://docs.stripe.com/api/invoices/object#invoice_object-lines).

Invoice Items can be created before you are ready to actually send the invoice. This can be particularly useful when combined with a [subscription](https://docs.stripe.com/api/subscriptions). Sometimes you want to add a charge or credit to a customer, but actually charge or credit the customer's card only at the end of a regular billing cycle. This is useful for combining several charges (to minimize per-transaction fees), or for having Stripe tabulate your usage-based billing totals.

Related guides: [Integrate with the Invoicing API](https://docs.stripe.com/invoicing/integration), [Subscription Invoices](https://docs.stripe.com/billing/invoices/subscription#adding-upcoming-invoice-items).

You can now model subscriptions more flexibly using the [Prices API](#prices). It replaces the Plans API and is backwards compatible to simplify your migration.

Plans define the base price, currency, and billing cycle for recurring purchases of products. [Products](#products) help you track inventory or provisioning, and plans help you track pricing. Different physical goods or levels of service should be represented by products, and pricing options should be represented by plans. This approach lets you change prices without having to change your provisioning scheme.

For example, you might have a single “gold” product that has plans for $10/month, $100/year, €9/month, and €90/year.

Related guides: [Set up a subscription](https://docs.stripe.com/billing/subscriptions/set-up-subscription) and more about [products and prices](https://docs.stripe.com/products-prices/overview).

A Quote is a way to model prices that you’d like to provide to a customer. Once accepted, it will automatically create an invoice, subscription or subscription schedule.

Subscription items allow you to create customer subscriptions with more than one plan, making it easy to represent complex billing relationships.

A subscription schedule allows you to create and manage the lifecycle of a subscription by predefining expected changes.

Related guide: [Subscription schedules](https://docs.stripe.com/billing/subscriptions/subscription-schedules)

A test clock enables deterministic control over objects in testmode. With a test clock, you can create objects at a frozen time in the past or future, and advance to a specific future time to observe webhooks and state changes. After the clock advances, you can either validate the current state of your scenario (and test your assumptions), change the current state of your scenario (and test more complex scenarios), or keep advancing forward in time.

Usage records allow you to report customer usage and metrics to Stripe for metered billing of subscription prices.

Related guide: [Metered billing](https://docs.stripe.com/billing/subscriptions/metered-billing)

This is an object representing a Stripe account. You can retrieve it to see properties on the account like its current requirements or if the account is enabled to make live charges or receive payouts.

For Custom accounts, the properties below are always returned. For other accounts, some properties are returned until that account has started to go through Connect Onboarding. Once you create an [Account Link](https://docs.stripe.com/api/account_links) or [Account Session](https://docs.stripe.com/api/account_sessions), some properties are only returned for Custom accounts. Learn about the differences [between accounts](https://docs.stripe.com/connect/accounts).

Login Links are single-use login link for an Express account to access their Stripe dashboard.

Account Links are the means by which a Connect platform grants a connected account permission to access Stripe-hosted applications, such as Connect Onboarding.

Related guide: [Connect Onboarding](https://docs.stripe.com/connect/custom/hosted-onboarding)

An AccountSession allows a Connect platform to grant access to a connected account in Connect embedded components.

We recommend that you create an AccountSession each time you need to display an embedded component to your user. Do not save AccountSessions to your database as they expire relatively quickly, and cannot be used more than once.

Related guide: [Connect embedded components](https://docs.stripe.com/connect/get-started-connect-embedded-components)

When you collect a transaction fee on top of a charge made for your user (using [Connect](https://docs.stripe.com/connect)), an `Application Fee` object is created in your account. You can list, retrieve, and refund application fees.

Related guide: [Collecting application fees](https://docs.stripe.com/connect/direct-charges#collecting-fees)

`Application Fee Refund` objects allow you to refund an application fee that has previously been created but not yet refunded. Funds will be refunded to the Stripe account from which the fee was originally collected.

Related guide: [Refunding application fees](https://docs.stripe.com/connect/destination-charges#refunding-app-fee)

This is an object representing a capability for a Stripe account.

Related guide: [Account capabilities](https://docs.stripe.com/connect/account-capabilities)

Stripe needs to collect certain pieces of information about each account created. These requirements can differ depending on the account's country. The Country Specs API makes these rules available to your integration.

You can also view the information from this API call as [an online guide](https://docs.stripe.com/connect/required-verification-information).

External bank accounts are financial accounts associated with a Stripe platform's connected accounts for the purpose of transferring funds to or from the connected account's Stripe balance.

External account cards are debit cards associated with a Stripe platform's connected accounts for the purpose of transferring funds to or from the connected accounts Stripe balance.

To top up your Stripe balance, you create a top-up object. You can retrieve individual top-ups, as well as list all top-ups. Top-ups are identified by a unique, random ID.

Related guide: [Topping up your platform account](https://docs.stripe.com/connect/top-ups)

A `Transfer` object is created when you move funds between Stripe accounts as part of Connect.

Before April 6, 2017, transfers also represented movement of funds from a Stripe account to a card or bank account. This behavior has since been split out into a [Payout](#payout_object) object, with corresponding payout endpoints. For more information, read about the [transfer/payout split](https://docs.stripe.com/transfer-payout-split).

Related guide: [Creating separate charges and transfers](https://docs.stripe.com/connect/separate-charges-and-transfers)

[Stripe Connect](https://docs.stripe.com/connect) platforms can reverse transfers made to a connected account, either entirely or partially, and can also specify whether to refund any related application fees. Transfer reversals add to the platform's balance and subtract from the destination account's balance.

Reversing a transfer that was made for a [destination charge](https://docs.stripe.com/connect/destination-charges) is allowed only up to the

amount of the charge. It is possible to reverse a [transfer_group](https://docs.stripe.com/connect/separate-charges-and-transfers#transfer-options) transfer only if the destination account has enough balance to cover the reversal.

Related guide: [Reversing transfers](https://docs.stripe.com/connect/separate-charges-and-transfers#reversing-transfers)

Secret Store is an API that allows Stripe Apps developers to securely persist secrets for use by UI Extensions and app backends.

The primary resource in Secret Store is a `secret`. Other apps can't view secrets created by an app. Additionally, secrets are scoped to provide further permission control.

All Dashboard users and the app backend share `account` scoped secrets. Use the `account` scope for secrets that don't change per-user, like a third-party API key.

A `user` scoped secret is accessible by the app backend and one specific Dashboard user. Use the `user` scope for per-user secrets like per-user OAuth tokens, where different users might have different permissions.

Related guide: [Store data between page reloads](https://docs.stripe.com/stripe-apps/store-auth-data-custom-objects)

An early fraud warning indicates that the card issuer has notified us that a charge may be fraudulent.

Related guide: [Early fraud warnings](https://docs.stripe.com/disputes/measuring#early-fraud-warnings)

Reviews can be used to supplement automated fraud detection with human expertise.

Learn more about [Radar](https://docs.stripe.com/radar) and reviewing payments [here](https://docs.stripe.com/radar/reviews).

Value lists allow you to group values together which can then be referenced in rules.

Related guide: [Default Stripe lists](https://docs.stripe.com/radar/lists#managing-list-items)

Value list items allow you to add specific values to a given Radar value list, which can then be used in rules.

Related guide: [Managing list items](https://docs.stripe.com/radar/lists#managing-list-items)

As a [card issuer](https://docs.stripe.com/issuing), you can dispute transactions that the cardholder does not recognize, suspects to be fraudulent, or has other issues with.

Related guide: [Issuing disputes](https://docs.stripe.com/issuing/purchases/disputes)

Funding Instructions contain reusable bank account and routing information. Push funds to these addresses via bank transfer to [top up Issuing Balances](https://docs.stripe.com/issuing/funding/balance).

Any use of an [issued card](https://docs.stripe.com/issuing) that results in funds entering or leaving your Stripe account, such as a completed purchase or refund, is represented by an Issuing `Transaction` object.

Related guide: [Issued card transactions](https://docs.stripe.com/issuing/purchases/transactions)

A Connection Token is used by the Stripe Terminal SDK to connect to a reader.

Related guide: [Fleet management](https://docs.stripe.com/terminal/fleet/locations)

A TerminalHardwareOrder represents an order for Terminal hardware, containing information such as the price, shipping information and the items ordered.

A TerminalHardwareProduct is a category of hardware devices that are generally similar, but may have variations depending on the country it’s shipped to.

TerminalHardwareSKUs represent variations within the same Product (for example, a country specific device). For example, WisePOS E is a TerminalHardwareProduct and a WisePOS E – US and WisePOS E – UK are TerminalHardwareSKUs.

A TerminalHardwareSKU represents a SKU for Terminal hardware. A SKU is a representation of a product available for purchase, containing information such as the name, price, and images.

A TerminalHardwareShipping represents a Shipping Method for Terminal hardware. A Shipping Method is a country-specific representation of a way to ship hardware, containing information such as the country, name, and expected delivery date.

A Configurations object represents how features should be configured for terminal readers.

Stripe Treasury provides users with a container for money called a FinancialAccount that is separate from their Payments balance. FinancialAccounts serve as the source and destination of Treasury’s money movement APIs.

Encodes whether a FinancialAccount has access to a particular Feature, with a `status` enum and associated `status_details`. Stripe or the platform can control Features via the requested field.

TransactionEntries represent individual units of money movements within a single [Transaction](#transactions).

Use OutboundTransfers to transfer funds from a [FinancialAccount](#financial_accounts) to a PaymentMethod belonging to the same entity. To send funds to a different party, use [OutboundPayments](#outbound_payments) instead. You can send funds over ACH rails or through a domestic wire transfer to a user’s own external bank account.

Simulate OutboundTransfer state changes with the `/v1/test_helpers/treasury/outbound_transfers` endpoints. These methods can only be called on test mode objects.

Use OutboundPayments to send funds to another party’s external bank account or [FinancialAccount](#financial_accounts). To send money to an account belonging to the same user, use an [OutboundTransfer](#outbound_transfers).

Simulate OutboundPayment state changes with the `/v1/test_helpers/treasury/outbound_payments` endpoints. These methods can only be called on test mode objects.

ReceivedCredits represent funds sent to a [FinancialAccount](#financial_accounts) (for example, via ACH or wire). These money movements are not initiated from the FinancialAccount.

ReceivedDebits represent funds pulled from a [FinancialAccount](#financial_accounts). These are not initiated from the FinancialAccount.

You can reverse some [ReceivedCredits](#received_credits) depending on their network and source flow. Reversing a ReceivedCredit leads to the creation of a new object known as a CreditReversal.

You can reverse some [ReceivedDebits](#received_debits) depending on their network and source flow. Reversing a ReceivedDebit leads to the creation of a new object known as a DebitReversal.

If you have [scheduled a Sigma query](https://docs.stripe.com/sigma/

scheduled-queries), you'll receive a `sigma.scheduled_query_run.created` webhook each time the query runs. The webhook contains a `ScheduledQueryRun` object, which you can use to retrieve the query results.

The Report Run object represents an instance of a report type generated with specific run parameters. Once the object is created, Stripe begins processing the report. When the report has finished running, it will give you a reference to a file where you can retrieve your results. For an overview, see [API Access to Reports](https://docs.stripe.com/reporting/statements/api).

Note that certain report types can only be run based on your live-mode data (not test-mode data), and will error when queried without a [live-mode API key](https://docs.stripe.com/keys#test-live-modes).

The Report Type resource corresponds to a particular type of report, such as the "Activity summary" or "Itemized payouts" reports. These objects are identified by an ID belonging to a set of enumerated values. See [API Access to Reports documentation](https://docs.stripe.com/reporting/statements/api) for those Report Type IDs, along with required and optional parameters.

Note that certain report types can only be run based on your live-mode data (not test-mode data), and will error when queried without a [live-mode API key](https://docs.stripe.com/keys#test-live-modes).

A Financial Connections Account represents an account that exists outside of Stripe, to which you have been granted some degree of access.

Describes an owner of an account.

A Financial Connections Session is the secure way to programmatically launch the client-side Stripe.js modal that lets your users link their accounts.

A Transaction represents a real transaction that affects a Financial Connections Account balance.

A Tax `Registration` lets us know that your business is registered to collect tax on payments within a region, enabling you to [automatically collect tax](https://docs.stripe.com/tax).

Stripe doesn't register on your behalf with the relevant authorities when you create a Tax `Registration` object. For more information on how to register to collect tax, see [our guide](https://docs.stripe.com/tax/registering).

Related guide: [Using the Registrations API](https://docs.stripe.com/

tax/registrations-api)

You can use Tax `Settings` to manage configurations used by Stripe Tax calculations.

Related guide: [Using the Settings API](https://docs.stripe.com/tax/settings-api)

A VerificationSession guides you through the process of collecting and verifying the identities of your users. It contains details about the type of verification, such as what [verification check](https://docs.stripe.com/identity/verification-checks) to perform. Only create one VerificationSession for each verification in your system.

A VerificationSession transitions through [multiple statuses](https://docs.stripe.com/identity/how-sessions-work) throughout its lifetime as it progresses through the verification flow. The VerificationSession contains the user’s verified data after verification checks are complete.

Related guide: [The Verification Sessions API](https://docs.stripe.com/identity/verification-sessions)

A VerificationReport is the result of an attempt to collect and verify data from a user. The collection of verification checks performed is determined from the `type` and `options` parameters used. You can find the result of each verification check performed in the appropriate sub-resource: `document`, `id_number`, `selfie`.

Each VerificationReport contains a copy of any data collected by the user as well as reference IDs which can be used to access collected images through the [FileUpload](https://docs.stripe.com/api/files) API. To configure and create VerificationReports, use the [VerificationSession](https://docs.stripe.com/api/identity/verification_sessions) API.

Related guides: [Accessing verification results](https://docs.stripe.com/identity/verification-sessions#results).

A Crypto Onramp Session represents your customer’s session as they purchase cryptocurrency through Stripe. Once payment is successful, Stripe will fulfill the delivery of cryptocurrency to your user’s wallet and contain a reference to the crypto transaction ID.

You can create an onramp session on your server and embed the widget on your frontend. Alternatively, you can redirect your users to the standalone hosted onramp.

Related guide: [Integrate the onramp](https://docs.stripe.com/crypto/integrate-the-onramp)

Crypto Onramp Quotes are estimated quotes for onramp conversions into all the different cryptocurrencies on different networks. The Quotes API allows you to display quotes in your product UI before directing the user to the onramp widget.

Related guide: [Quotes API](https://docs.stripe.com/crypto/quotes-api)

Orders represent your intent to purchase a particular Climate product. When you create an order, the payment is deducted from your merchant balance.

A Climate product represents a type of carbon removal unit available for reservation. You can retrieve it to see the current price and availability.

A supplier of carbon removal.

Instructs Stripe to make a request on your behalf using the destination URL and HTTP method in the config. A config is set up for each destination URL by Stripe at the time of onboarding. Stripe verifies requests with your credentials in the config, and injects card details from the payment_method into the request.

Stripe redacts all sensitive fields and headers, including authentication credentials and card numbers, before storing the request and response data in the forwarding Request object, which are subject to a 30-day retention period.

You can provide a Stripe idempotency key to make sure that requests with the same key result in only one outbound request. The Stripe idempotency key provided should be unique and different from any idempotency keys provided on the underlying third-party request.

Forwarding Requests are synchronous requests that return a response or time out according to Stripe’s limits.

You can configure [webhook endpoints](https://docs.stripe.com/webhooks/) via the API to be notified about events that happen in your Stripe account or connected accounts.

Most users configure webhooks from [the dashboard](https://dashboard.stripe.com/webhooks), which provides a user interface for registering and testing your webhook endpoints.

Related guide: [Setting up webhooks](https://docs.stripe.com/webhooks/configure)