May 18, 2015 Rev 1.3
Peter Gustafson
Target link: https://stripe.com/docs/api#intro

API Reference
The Stripe API is organized around REST. Our API has predictable, resource-oriented URLs, and uses HTTP response codes to indicate API errors. We use built-in HTTP features, like HTTP authentication and HTTP verbs, which are understood by off-the-shelf HTTP clients. We support cross-origin resource sharing, allowing you to interact securely with our API from a client-side web application (though you should never expose your secret API key in any public website's client-side code). JSON is returned by all API responses, including errors, although our API libraries convert responses to appropriate language-specific objects.

To make the API as explorable as possible, accounts have test mode and live mode API keys. There is no "switch" for changing between modes, just use the appropriate key to perform a live or test transaction. Requests made with test mode credentials never hit the banking networks and incur no cost.

We send information on new additions and changes to Stripe's API and language libraries to the API announce mailing list. Be sure to subscribe to stay informed.

The requests in the right sidebar are designed to work as is. The sample requests are performed using a test mode API key, sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ, linked to your account under the email address peterdavidgustafson@gmail.com. Only you can see these account-specific values.

API libraries
Official libraries for the Stripe API are available in several languages. Community-supported libraries are also available for additional languages.

https://api.stripe.com
Authentication
Authenticate your account when using the API by including your secret API key in the request. You can manage your API keys in the Dashboard. Your API keys carry many privileges, so be sure to keep them secret! Do not share your secret API keys in publicly accessible areas such GitHub, client-side code, and so forth.

To use your API key, assign it to Stripe.api_key. The Ruby library will then automatically send this key in each request.

All API requests must be made over HTTPS. Calls made over plain HTTP will fail. API requests without authentication will also fail.

```
require "stripe"
Stripe.api_key = "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"
```
You can also set a per-request key like in the example below. This is often useful for Connect applications that use multiple API keys during the lifetime of a process.

Authentication is transparently handled for you in subsequent method calls on the returned object.

```
Stripe::Charge.retrieve(
  "ch_1BqnFrBMHHbTYGW2twbHvJrO",
  :api_key => "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"
)
```
Your test API key is included in all the examples on this page, so you can test any example right away. Only you can see this value.

Errors

Stripe uses conventional HTTP response codes to indicate the success or failure of an API request. In general, codes in the 2xx range indicate success, codes in the 4xx range indicate an error that failed given the information provided (e.g., a required parameter was omitted, a charge failed, etc.), and codes in the 5xx range indicate an error with Stripe's servers (these are rare).

Not all errors map cleanly onto HTTP response codes, however. When a request is valid but does not complete successfully (e.g., a card is declined), we return a 402 error code. To understand why a card is declined, refer to the list of codes in the documentation.

ATTRIBUTES

type

The type of error returned. Can be: api_connection_error, api_error, authentication_error, card_error, idempotency_error invalid_request_error, or rate_limit_error.

charge

The ID of the failed charge.

message

optional

A human-readable message providing more details about the error. For card errors, these messages can be shown to your users.

code

optional

For card errors, a short string from among those listed on the right describing the kind of card error that occurred.

decline_code
optional
For card errors resulting from a card issuer decline, a short string indicating the card issuer's reason for the decline if they provide one.

param
optional
The parameter the error relates to if the error is parameter-specific. You can use this to display a message near the correct form field, for example.

HTTP status code summary
200 - OK Everything worked as expected.
400 - Bad Request The request was unacceptable, often due to missing a required parameter.
401 - Unauthorized No valid API key provided.
402 - Request Failed The parameters were valid but the request failed.
404 - Not Found The requested resource doesn't exist.
409 - Conflict The request conflicts with another request (perhaps due to using the same idempotent key).
429 - Too Many Requests Too many requests hit the API too quickly. We recommend an exponential backoff of your requests.
500, 502, 503, 504 - Server Errors Something went wrong on Stripe's end. (These are rare.)
Errors
TYPES
api_connection_error Failure to connect to Stripe's API.
api_error API errors cover any other type of problem (e.g., a temporary problem with Stripe's servers) and are extremely uncommon.
authentication_error Failure to properly authenticate yourself in the request.
card_error Card errors are the most common type of error you should expect to handle. They result when the user enters a card that can't be charged for some reason.
idempotency_error Idempotency errors occur when an Idempotency-Key is re-used on a request that does not match the API endpoint and parameters of the first.
invalid_request_error Invalid request errors arise when your request has invalid parameters.
rate_limit_error Too many requests hit the API too quickly.
validation_error Errors triggered by our client-side libraries when failing to validate fields (e.g., when a card number or expiration date is invalid or incomplete).
CODES
invalid_number The card number is not a valid credit card number.
invalid_expiry_month The card's expiration month is invalid.
invalid_expiry_year The card's expiration year is invalid.

invalid_cvc The card's security code is invalid.
invalid_swipe_data The card's swipe data is invalid.
incorrect_number The card number is incorrect.
expired_card The card has expired.
incorrect_cvc The card's security code is incorrect.
incorrect_zip The card's zip code failed validation.
card_declined The card was declined.
missing There is no card on a customer that is being charged.
processing_error An error occurred while processing the card.
Radar provides built-in rules for CVC and ZIP validation that can be enabled/disabled in the Dashboard.
Handling errors
Our API libraries raise exceptions for many reasons, such as a failed charge, invalid parameters, authentication errors, and network unavailability. We recommend writing code that gracefully handles all possible API exceptions.

```
begin
  # Use Stripe's library to make requests...
rescue Stripe::CardError => e
  # Since it's a decline, Stripe::CardError will be caught
  body = e.json_body
  err  = body[:error]

  puts "Status is: #{e.http_status}"
  puts "Type is: #{err[:type]}"
  puts "Charge ID is: #{err[:charge]}"
  # The following fields are optional
  puts "Code is: #{err[:code]}" if err[:code]
  puts "Decline code is: #{err[:decline_code]}" if err[:decline_code]
  puts "Param is: #{err[:param]}" if err[:param]
  puts "Message is: #{err[:message]}" if err[:message]
rescue Stripe::RateLimitError => e
  # Too many requests made to the API too quickly
rescue Stripe::InvalidRequestError => e
  # Invalid parameters were supplied to Stripe's API
rescue Stripe::AuthenticationError => e
  # Authentication with Stripe's API failed
  # (maybe you changed API keys recently)
rescue Stripe::APIConnectionError => e
  # Network communication with Stripe failed
rescue Stripe::StripeError => e
  # Display a very generic error to the user, and maybe send
  # yourself an email
```

```
rescue => e
  # Something else happened, completely unrelated to Stripe
end
```

Expanding Objects

Many objects contain the ID of a related object in their response properties. For example, a Charge may have an associated Customer ID. Those objects can be expanded inline with the expand request parameter. Objects that can be expanded are noted in this documentation. This parameter is available on all API requests, and applies to the response of that request only.

You can nest expand requests with the dot property. For example, requesting invoice.customer on a charge will expand the invoice property into a full Invoice object, and will then expand the customer property on that invoice into a full Customer object.

You can expand multiple objects at once by identifying multiple items in the expand array.

```
require "stripe"
Stripe.api_key = "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"

Stripe::Charge.retrieve({ :id => "ch_1BqnFsBMHHbTYGW2REkSnJGd", :expand => ['customer']
})
```

Idempotent Requests

The API supports idempotency for safely retrying requests without accidentally performing the same operation twice. For example, if a request to create a charge fails due to a network connection error, you can retry the request with the same idempotency key to guarantee that only a single charge is created.

GET and DELETE requests are idempotent by definition, meaning that the same backend work will occur no matter how many times the same request is issued. You shouldn't send an idempotency key with these verbs because it will have no effect.

To perform an idempotent request, provide an additional idempotency_key element to the request options.

How you create unique keys is up to you, but we suggest using V4 UUIDs or another appropriately random string. We'll always send back the same response for requests made with the same key, and keys can't be reused with different request parameters. Keys expire after 24 hours.

```
require "stripe"
Stripe.api_key = "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"

Stripe::Charge.create({
  :amount => 2000,
```

```
  :currency => "usd",
  :source => "tok_visa", # obtained with Stripe.js
  :description => "Charge for andrew.white@example.com"
}, {
  :idempotency_key => "mGWExZMddBG2uxXK"
})
```

Metadata

Updatable Stripe objects—including Account, Charge, Customer, Refund, Subscription, and Transfer—have a metadata parameter. You can use this parameter to attach key-value data to these Stripe objects.

Metadata is useful for storing additional, structured information on an object. As an example, you could store your user's full name and corresponding unique identifier from your system on a Stripe Customer object. Metadata is not used by Stripe (e.g., to authorize or decline a charge), and won't be seen by your users unless you choose to show it to them.

Some of the objects listed above also support a description parameter. You can use the description parameter to annotate a charge, for example, with a human-readable description, such as "2 shirts for test@example.com". Unlike metadata, description is a single string, and your users may see it (e.g., in email receipts Stripe sends on your behalf).

Note: You can specify up to 20 keys, with key names up to 40 characters long and values up to 500 characters long.

SAMPLE METADATA USE CASES

Link IDs

Attach your system's unique IDs to a Stripe object for easy lookups. Add your order number to a charge, your user ID to a customer or recipient, or a unique receipt number to a transfer, for example.

Refund papertrails

Store information about why a refund was created, and by whom.

Customer details

Annotate a customer by storing the customer's phone number for your later use.

```
 require "stripe"
Stripe.api_key = "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"

Stripe::Charge.create(
  :amount => 2000,
  :currency => "usd",
  :source => "tok_amex", # obtained with Stripe.js
```

```
  :metadata => {'order_id' => '6735'}
)
 #<Stripe::Charge id=ch_1BqnFsBMHHbTYGW2REkSnJGd 0x00000a> JSON: {
  "id": "ch_1BqnFsBMHHbTYGW2REkSnJGd",
  "object": "charge",
  "amount": 100,
  "amount_refunded": 0,
  "application": null,
  "application_fee": null,
  "balance_transaction": "txn_1BqnFsBMHHbTYGW2q5vudPsg",
  "captured": false,
  "created": 1517511472,
  "currency": "usd",
  "customer": null,
  "description": "My First Test Charge (created for API docs)",
  "destination": null,
  "dispute": null,
  "failure_code": null,
  "failure_message": null,
  "fraud_details": {
  },
  "invoice": null,
  "livemode": false,
  "metadata": {
    "order_id": "6735"
  },
  "on_behalf_of": null,
  "order": null,
  "outcome": null,
  "paid": true,
  "receipt_email": null,
  "receipt_number": null,
  "refunded": false,
  "refunds": {
    "object": "list",
    "data": [

    ],
    "has_more": false,
    "total_count": 0,
    "url": "/v1/charges/ch_1BqnFsBMHHbTYGW2REkSnJGd/refunds"
  },
  "review": null,
```

```
  "shipping": null,
  "source": {
    "id": "card_19QnerBMHHbTYGW2nxgDxBzR",
    "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": null,
    "cvc_check": null,
    "dynamic_last4": null,
    "exp_month": 8,
    "exp_year": 2017,
    "fingerprint": "DuWe4kcGPrRJnhC5",
    "funding": "credit",
    "last4": "4242",
    "metadata": {
    },
    "name": null,
    "tokenization_method": null
  },
  "source_transfer": null,
  "statement_descriptor": null,
  "status": "succeeded",
  "transfer_group": null
}
```


Pagination

All top-level API resources have support for bulk fetches via "list" API methods. For instance you can list charges, list customers, and list invoices. These list API methods share a common structure, taking at least these three parameters: limit, starting_after, and ending_before.

Stripe utilizes cursor-based pagination via the starting_after and ending_before parameters. Both take an existing object ID value (see below) and return objects in reverse chronological order. The ending_before parameter returns objects listed before the named object. The starting_after parameter returns objects listed after the named object. If both parameters are provided, only ending_before is used.

ARGUMENTS

limit

optional, default is 10

A limit on the number of objects to be returned, between 1 and 100.

starting_after

optional

A cursor for use in pagination. starting_after is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include starting_after=obj_foo in order to fetch the next page of the list.

ending_before

optional

A cursor for use in pagination. ending_before is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, starting with obj_bar, your subsequent call can include ending_before=obj_bar in order to fetch the previous page of the list.

LIST RESPONSE FORMAT

object

string, value is "list"

A string describing the object type returned.

data

array

An array containing the actual response elements, paginated by any request parameters.

has_more

boolean

Whether or not there are more elements available after this set. If false, this set comprises the end of the list.

url

string

The URL for accessing this list.

```
require "stripe"
Stripe.api_key = "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"

Stripe::Customer.list(limit: 3)
#<Stripe::ListObject:0x3fe634d74498> JSON: {
  "object": "list",
  "url": "/v1/customers",
```

```
  "has_more": false,
  "data": [
    #<Stripe::Customer id=cus_CFHpNCc0DP4eSv 0x00000a> JSON: {
      "id": "cus_CFHpNCc0DP4eSv",
      "object": "customer",
      "account_balance": 0,
      "created": 1517511472,
      "currency": "usd",
      "default_source": null,
      "delinquent": false,
      "description": null,
      "discount": null,
      "email": null,
      "livemode": false,
      "metadata": {
      },
      "shipping": null,
      "sources": {
        "object": "list",
        "data": [

        ],
        "has_more": false,
        "total_count": 0,
        "url": "/v1/customers/cus_CFHpNCc0DP4eSv/sources"
      },
      "subscriptions": {
        "object": "list",
        "data": [

        ],
        "has_more": false,
        "total_count": 0,
        "url": "/v1/customers/cus_CFHpNCc0DP4eSv/subscriptions"
      }
    },
    #<Stripe::Customer[...] ...>,
    #<Stripe::Customer[...] ...>
  ]
}
```


Auto-pagination

Most of our libraries support auto-pagination. This feature easily handles fetching large lists of resources without having to manually paginate results and perform subsequent requests.

To use the auto-pagination feature in Ruby, simply issue an initial "list" call with the parameters you need, then call auto_paging_each on the returned list object to iterate over all objects matching your initial parameters.

```
require "stripe"
Stripe.api_key = "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"

customers = Stripe::Customer.all(:limit => 3)
customers.auto_paging_each do |customer|
  # Do something with customer
end
```

Request IDs

Each API request has an associated request identifier. You can find this value in the response headers, under Request-Id. You can also find request identifiers in the URLs of individual request logs in your Dashboard. If you need to contact us about a specific request, providing the request identifier will ensure the fastest possible resolution.

Versioning

When we make backwards-incompatible changes to the API, we release new, dated versions. You are running the current version of the API, 2018-01-23. Read our API upgrades guide to see our API changelog and to learn more about backwards compatibility.

All requests will use your account API settings, unless you override the API version. The changelog lists every available version. Note that events generated by API requests will always be structured according to your account API version.

To override the API version, assign the version to the Stripe.api_version property.

You can visit your Dashboard to upgrade your API version. As a precaution, use API versioning to test a new API version before committing to an upgrade.

```
require "stripe"
Stripe.api_key = "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"
Stripe.api_version = "2018-01-23"
```

Balance

This is an object representing your Stripe balance. You can retrieve it to see the balance currently on your Stripe account.

You can also retrieve a list of the balance history, which contains a list of transactions that contributed to the balance (e.g., charges, payouts, and so forth).

The available and pending amounts for each currency are broken down further by payment source types.

The balance object
ATTRIBUTES
object
string , value is "balance"
String representing the object’s type. Objects of the same type share the same value.

available
array
Funds that are available to be paid out automatically by Stripe or explicitly via the transfers API. The available balance for each currency and payment type can be found in the source_types property.

connect_reserved
array
Funds held due to negative balances on connected Custom accounts. The connect reserve balance for each currency and payment type can be found in the source_types property.

livemode
boolean
Flag indicating whether the object exists in live mode or test mode.

pending
array
Funds that are not available in the balance yet, due to the 7-day rolling pay cycle. The pending balance for each currency and payment type can be found in the source_types property.

#<Stripe::Balance 0x00000a> JSON: {
  "object": "balance",
  "available": [
    {
      "currency": "usd",
      "amount": 0,
      "source_types": {
        "card": 0
      }
    }
  ],
  "livemode": false,
  "pending": [
    {

```
      "currency": "usd",
      "amount": 0,
      "source_types": {
        "card": 0
      }
    }
  ]
}
```


The balance_transaction object

ATTRIBUTES

id
string
Unique identifier for the object.

object
string , value is "balance_transaction"
String representing the object’s type. Objects of the same type share the same value.

amount
integer
Gross amount of the transaction, in cents.

available_on
timestamp
The date the transaction’s net funds will become available in the Stripe balance.

created
timestamp
Time at which the object was created. Measured in seconds since the Unix epoch.

currency
currency
Three-letter ISO currency code, in lowercase. Must be a supported currency.

description
string
An arbitrary string attached to the object. Often useful for displaying to users.

exchange_rate
decimal

fee
integer
Fees (in cents) paid for this transaction.

fee_details
list
Detailed breakdown of fees (in cents) paid for this transaction.

net
integer
Net amount of the transaction, in cents.

source
string
The Stripe object this transaction is related to.

status
string
If the transaction’s net funds are available in the Stripe balance yet. Either available or pending.

type
string
Transaction type: adjustment, application_fee, application_fee_refund, charge, payment, payment_failure_refund, payment_refund, refund, transfer, transfer_refund, payout, payout_cancel, payout_failure, validation, or stripe_fee.

#<Stripe::BalanceTransaction id=txn_1BqnFsBMHHbTYGW2IlgXof07 0x00000a> JSON: {
  "id": "txn_1BqnFsBMHHbTYGW2IlgXof07",
  "object": "balance_transaction",
  "amount": 100,
  "available_on": 1517511472,
  "created": 1517511472,
  "currency": "usd",
  "description": null,
  "exchange_rate": null,
  "fee": 0,
  "fee_details": [

  ],
  "net": 100,
  "source": "ch_1BqnFsBMHHbTYGW2cg6LysET",
  "status": "pending",
  "type": "charge"
}
Retrieve balance

Retrieves the current account balance, based on the authentication that was used to make the request.

ARGUMENTS
No arguments…
Returns
Returns a balance object for the account authenticated as.

```
 require "stripe"
Stripe.api_key = "sk_test_qMgXKP8I1TZDUCSKV7PN5YZQ"

Stripe::Balance.retrieve()
 #<Stripe::Balance 0x00000a> JSON: {
  "object": "balance",
  "available": [
    {
      "currency": "usd",
      "amount": 0,
      "source_types": {
        "card": 0
      }
    }
  ],
  "livemode": false,
  "pending": [
    {
      "currency": "usd",
      "amount": 0,
      "source_types": {
        "card": 0
      }
    }
  ]
}
```

Retrieve a balance transaction
Retrieves the balance transaction with the given ID.

ARGUMENTS
 id
REQUIRED
The ID of the desired balance transaction (as found on any API object that affects the balance, e.g., a charge or transfer).