API Reference

The Stripe API is organized around REST. Our API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.

You can use the Stripe API in test mode, which doesn't affect your live data or interact with the banking networks. The API key you use to authenticate the request determines whether the request is live mode or test mode.

The Stripe API doesn't support bulk updates. You can work on only one object per request.

The Stripe API differs for every account as we release new versions and tailor functionality. Log in to see docs customized to your version of the API, with your test key and data.

Authentication

The Stripe API uses API keys to authenticate requests. You can view and manage your API keys in the Stripe Dashboard.

Test mode secret keys have the prefix `sk_test_` and live mode secret keys have the prefix `sk_live_`. Alternatively, you can use restricted API keys for granular permissions.

Your API keys carry many privileges, so be sure to keep them secure! Do not share your secret API keys in publicly accessible areas such as GitHub, client-side code, and so forth.

Authentication to the API is performed via HTTP Basic Auth. Provide your API key as the basic auth username value. You do not need to provide a password.

If you need to authenticate via bearer auth (e.g., for a cross-origin request), use `-H "Authorization: Bearer sk_test_4eC39HqLyjWDarjtT1zdp7dc"` instead of `-u sk_test_4eC39HqLyjWDarjtT1zdp7dc`.

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

Related video: Authentication.

Connected Accounts

Clients can make requests as connected accounts using the special header `Stripe-Account` which should contain a Stripe account ID (usually starting with the prefix `acct_`).

See also Making API calls for connected accounts.

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.). Codes in the `5xx` range indicate an error with Stripe's servers (these are rare).

Some `4xx` errors that could be handled programmatically (e.g., a card is declined) include an error code that briefly explains the error reported.

Attributes

`type` string
The type of error returned. One of `api_error`, `card_error`, `idempotency_error`, or `invalid_request_error`

`code` string
For some errors that could be handled programmatically, a short string indicating the error code reported.

`decline_code` string
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.

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

`param` string
If the error is parameter-specific, the parameter related to the error. For example, you can use this to display a message near the correct form field.

`payment_intent` hash
The PaymentIntent object for errors returned on a request involving a PaymentIntent.

+ Show child attributes

More attributes

Expand all

- **charge** string
- **doc_url** string
- **payment_method** hash
- **payment_method_type** string
- **request_log_url** string
- **setup_intent** hash
- **source** hash

Handling errors

Our Client 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.

Related guide: Error Handling.

Expanding Responses

Many objects allow you to request additional information as an expanded response by using the `expand` request parameter. This parameter is available on all API requests, and applies to the response of that request only. Responses can be expanded in two ways.

In many cases, an object contains the ID of a related object in its response properties. For example, a `Charge` may have an associated `Customer` ID. Those objects can be expanded inline with the `expand` request parameter. ID fields that can be expanded into objects are noted in this documentation with the `expandable` label.

In some cases, such as the Issuing Card object's `number` and `cvc` fields, there are available fields that are not included in responses by default. You can request these fields as an expanded response by using the `expand` request parameter. Fields that can be included in an expanded response are noted in this documentation with the `expandable` label.

You can expand recursively by specifying nested fields after a dot (`.`). For example, requesting `invoice.subscription` on a charge will expand the `invoice` property into a full Invoice object, and will then expand the `subscription` property on that invoice into a full Subscription object.

You can use the `expand` param on any endpoint which returns expandable fields, including list, create, and update endpoints.

Expansions on list requests start with the `data` property. For example, you would expand `data.customers` on a request to list charges and associated customers. Many deep expansions on list requests can be slow.

Expansions have a maximum depth of four levels (so for example, when listing charges, `data.invoice.subscription.default_source` is the deepest allowed).

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

Related Guide: Expanding responses

Related video: Expand.

Idempotent Requests

The API supports idempotency for safely retrying requests without accidentally performing the same operation twice. When creating or updating an object, use an idempotency key. Then, if a connection error occurs, you can safely repeat the request without risk of creating a second object or performing the update twice.

To perform an idempotent request, provide an additional `Idempotency-Key: <key>` header to the request.

Stripe's idempotency works by saving the resulting status code and body of the first request made for any given idempotency key, regardless of whether it succeeded or failed. Subsequent requests with the same key return the same result, including `500` errors.

An idempotency key is a unique value generated by the client which the server uses to recognize subsequent retries of the same request. How you create unique keys is up to you, but we suggest using V4 UUIDs, or another random string with enough entropy to avoid collisions. Idempotency keys can be up to 255 characters long.

Keys are eligible to be removed from the system automatically after they're at least 24 hours old, and a new request is generated if a key is reused after the original has been pruned. The idempotency layer compares incoming parameters to those of the original request and errors unless they're the same to prevent accidental misuse.

Results are only saved if an API endpoint started executing. If incoming parameters failed validation, or the request conflicted with another that was executing concurrently, no idempotent result is saved because no API endpoint began execution. It is safe to retry these requests. For more information on when it is safe to retry idempotent requests, see this page.

All `POST` requests accept idempotency keys. Sending idempotency keys in `GET` and `DELETE` requests has no effect and should be avoided, as these requests are idempotent by definition.

Related video: Idempotency and retries.

Metadata

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

You can specify up to 50 keys, with key names up to 40 characters long and values up to 500 characters long.

Metadata is useful for storing additional, structured information on an object. For example, you could store your user's corresponding unique identifier from your system on a Stripe Customer object. By default, metadata isn't used by Stripe—for example, it's not used to authorize or decline a charge—but metadata is supported by the Search API and custom Radar rules. Your users won't see metadata unless you 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—with, for example, a human-readable description like `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).

Don't store any sensitive information (bank account numbers, card details, and so on) in metadata or in the `description` parameter.

Related video: Metadata.

Sample metadata use cases

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

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

Customer details
Annotate a customer by storing an internal ID for your later use.

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`.

The response of a list API method represents a single page in a reverse chronological stream of objects. If you do not specify `starting_after` or `ending_before`, you will receive the first page of this stream, containing the newest objects. You can specify `starting_after` equal to the object ID value (see below) of an item to retrieve the page of **older** objects occurring immediately **after** the named object in the reverse chronological stream. Similarly, you can specify `ending_before` to receive a page of **newer** objects occurring immediately **before** the named object in the stream. Objects in a page always appear in reverse chronological order. Only one of `starting_after` or `ending_before` may be used.

Our client libraries offer auto-pagination helpers to easily traverse all pages of a list.

Related video: Pagination and auto-pagination.

Parameters

`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.

Search

Some top-level API resource have support for retrieval via "search" API methods. For example, you can search charges, search customers, and search subscriptions.

Stripe's search API methods utilize cursor-based pagination via the `page` request parameter and `next_page` response parameter. For example, if you make a search request and receive `"next_page": "pagination_key"` in the response, your subsequent call can include `page=pagination_key` to fetch the next page of results.

Our client libraries offer auto-pagination helpers to easily traverse all pages of a search result.

Search request format

`query` REQUIRED

The search query string. See search query language.

`limit` optional

A limit on the number of objects to be returned. Limit can range between 1 and 100, and the default is 10.

page optional
A cursor for pagination across multiple pages of results. Don't include this parameter on the first call. Use the next_page value returned in a previous response to request subsequent results.

Search response format

object string, value is "search_result"
A string describing the object type returned.

url string
The URL for accessing this list.

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

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

next_page string
A cursor for use in pagination. If `has_more` is true, you can pass the value of `next_page` to a subsequent call to fetch the next page of results.

total_count optional positive integer or zero EXPANDABLE
The total number of objects that match the query, only accurate up to 10,000. This field is not included by default. To include it in the response, expand the `total_count` field.

Auto-pagination

Our libraries support auto-pagination. This feature allows you to easily iterate through large lists of resources without having to manually perform the requests to fetch subsequent pages.

Auto-paginating without `ending_before` will iterate in reverse chronological order. Auto-paginating with `ending_before` will iterate in forwards chronological order, despite the fact that the order of items in a list API response is always reverse chronological.

Since curl simply emits raw HTTP requests, it doesn't support auto-pagination.

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 backwards-incompatible changes are made to the API, a new, dated version is released. The current version is **2022-11-15**. Read our API upgrades guide to learn more about backwards compatibility. For all API updates, view our API changelog.

All requests use your account API settings, unless you override the API version. The upgrades guide lists every available version. *Note that by default webhook events are structured according to your account API version, unless you set an API version during endpoint creation.*

To set the API version on a specific request, send a `Stripe-Version` header.

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.

Related video: Versioning.

Subscriptions

Subscriptions allow you to charge a customer on a recurring basis.

Related guide: Creating Subscriptions.

The subscription object

Attributes

`id` string
Unique identifier for the object.

`cancel_at_period_end` boolean
If the subscription has been canceled with the `at_period_end` flag set to `true`, `cancel_at_period_end` on the subscription will be true. You can use this attribute to determine whether a subscription that has a status of active is scheduled to be canceled at the end of the current period.

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

`current_period_end` timestamp
End of the current period that the subscription has been invoiced for. At the end of this period, a new invoice will be created.

`current_period_start` timestamp
Start of the current period that the subscription has been invoiced for.

`customer` string EXPANDABLE
ID of the customer who owns the subscription.

`default_payment_method` string EXPANDABLE
ID of the default payment method for the subscription. It must belong to the customer associated with the subscription. This takes precedence over `default_source`. If neither are set, invoices will use the customer's invoice_settings.default_payment_method or default_source.

`description` string
The subscription's description, meant to be displayable to the customer. Use this field to optionally store an explanation of the subscription for rendering in Stripe surfaces.

`items` list
List of subscription items, each with an attached price.

+ Show child attributes

`latest_invoice` string EXPANDABLE
The most recent invoice this subscription has generated.

`metadata` hash

Set of key-value pairs that you can attach to an object. This can be useful for storing additional information about the object in a structured format.

`pending_setup_intent` string EXPANDABLE

You can use this SetupIntent to collect user authentication when creating a subscription without immediate payment or updating a subscription's payment method, allowing you to optimize for off-session payments. Learn more in the SCA Migration Guide.

`pending_update` hash

If specified, pending updates that will be applied to the subscription once the `latest_invoice` has been paid.

+ Show child attributes

`status` enum

Possible values are `incomplete`, `incomplete_expired`, `trialing`, `active`, `past_due`, `canceled`, or `unpaid`.

For `collection_method=charge_automatically` a subscription moves into `incomplete` if the initial payment attempt fails. A subscription in this state can only have metadata and default_source updated. Once the first invoice is paid, the subscription moves into an `active` state. If the first invoice is not paid within 23 hours, the subscription transitions to `incomplete_expired`. This is a terminal state, the open invoice will be voided and no further invoices will be generated.

A subscription that is currently in a trial period is `trialing` and moves to `active` when the trial period is over.

If subscription `collection_method=charge_automatically` it becomes `past_due` when payment to renew it fails and `canceled` or `unpaid` (depending on your subscriptions settings) when Stripe has exhausted all payment retry attempts.

If subscription `collection_method=send_invoice` it becomes `past_due` when its invoice is not paid by the due date, and `canceled` or `unpaid` if it is still not paid by an additional deadline after that. Note that when a subscription has a status of `unpaid`, no subsequent invoices will be attempted (invoices will be created, but then immediately automatically closed). After receiving updated payment information from a customer, you may choose to reopen and pay their closed invoices.

Possible enum values
`active`
`past_due`
`unpaid`
`canceled`

`incomplete`
`incomplete_expired`
`trialing`
`paused`

More attributes

Expand all

- **object** string, value is "subscription"
- **application** string EXPANDABLE "APPLICATION" CONNECT ONLY
- **application_fee_percent** decimal CONNECT ONLY
- **automatic_tax** hash
- **billing_cycle_anchor** timestamp
- **billing_thresholds** hash
- **cancel_at** timestamp
- **canceled_at** timestamp
- **cancellation_details** hash
- **collection_method** string
- **created** timestamp
- **days_until_due** integer
- **default_source** string EXPANDABLE BANK ACCOUNT, CARD, OR SOURCE (E.G., CARD)
- **default_tax_rates** array of hashes
- **discount** hash, discount object
- **ended_at** timestamp
- **livemode** boolean

- **next_pending_invoice_item_invoice** timestamp
- **on_behalf_of** string EXPANDABLE CONNECT ONLY
- **pause_collection** hash
- **payment_settings** hash
- **pending_invoice_item_interval** hash
- **schedule** string EXPANDABLE
- **start_date** timestamp
- **test_clock** string EXPANDABLE
- **transfer_data** hash CONNECT ONLY
- **trial_end** timestamp
- **trial_settings** hash
- **trial_start** timestamp

Create a subscription

Creates a new subscription on an existing customer. Each customer can have up to 500 active or scheduled subscriptions.

When you create a subscription with `collection_method=charge_automatically`, the first invoice is finalized as part of the request. The `payment_behavior` parameter determines the exact behavior of the initial payment.

To start subscriptions where the first invoice always begins in a `draft` status, use subscription schedules instead. Schedules provide the flexibility to model more complex billing configurations that change over time.

Parameters

customer REQUIRED

The identifier of the customer to subscribe.

`items` REQUIRED

A list of up to 20 subscription items, each with an attached price.

+ Show child parameters

`cancel_at_period_end` optional

Boolean indicating whether this subscription should cancel at the end of the current period.

`currency` optional

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

`default_payment_method` optional

ID of the default payment method for the subscription. It must belong to the customer associated with the subscription. This takes precedence over `default_source`. If neither are set, invoices will use the customer's invoice_settings.default_payment_method or default_source.

`description` optional

The subscription's description, meant to be displayable to the customer. Use this field to optionally store an explanation of the subscription for rendering in Stripe surfaces.

`metadata` optional dictionary

Set of key-value pairs 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`.

`payment_behavior` optional enum

Only applies to subscriptions with `collection_method=charge_automatically`.

Use `allow_incomplete` to create subscriptions with `status=incomplete` if the first invoice cannot be paid. Creating subscriptions with this status allows you to manage scenarios where additional user actions are needed to pay a subscription's invoice. For example, SCA regulation may require 3DS authentication to complete payment. See the SCA Migration Guide for Billing to learn more. This is the default behavior.

Use `default_incomplete` to create Subscriptions with `status=incomplete` when the first invoice requires payment, otherwise start as active. Subscriptions transition to `status=active` when successfully confirming the payment intent on the first invoice. This allows simpler management of scenarios where additional user actions are needed to pay a subscription's invoice. Such as failed payments, SCA regulation, or collecting a mandate for a bank debit payment method. If the payment intent is not confirmed within 23 hours subscriptions transition to `status=incomplete_expired`, which is a terminal state.

Use `error_if_incomplete` if you want Stripe to return an HTTP 402 status code if a subscription's first invoice cannot be paid. For example, if a payment method requires 3DS authentication due to SCA regulation and further user action is needed, this parameter does not create a subscription and returns an error instead. This was the default behavior for API versions prior to 2019-03-14. See the changelog to learn more.

`pending_if_incomplete` is only used with updates and cannot be passed when creating a subscription.

Subscriptions with `collection_method=send_invoice` are automatically activated regardless of the first invoice status.

Possible enum values
`allow_incomplete`
`error_if_incomplete`
`pending_if_incomplete`
`default_incomplete`

More parameters

Expand all

- **`add_invoice_items`** optional array of hashes
- **`application_fee_percent`** optional CONNECT ONLY
- **`automatic_tax`** optional dictionary
- **`backdate_start_date`** optional
- **`billing_cycle_anchor`** optional
- **`billing_thresholds`** optional dictionary
- **`cancel_at`** optional
- **`collection_method`** optional
- **`coupon`** optional
- **`days_until_due`** optional
- **`default_source`** optional

- **default_tax_rates** optional
- **off_session** optional
- **on_behalf_of** optional
- **payment_settings** optional dictionary
- **pending_invoice_item_interval** optional dictionary
- **promotion_code** optional
- **proration_behavior** optional enum
- **transfer_data** optional dictionary CONNECT ONLY
- **trial_end** optional
- **trial_from_plan** optional
- **trial_period_days** optional
- **trial_settings** optional dictionary

Returns

The newly created `Subscription` object, if the call succeeded. If the attempted charge fails, the subscription is created in an `incomplete` status.

Retrieve a subscription

Retrieves the subscription with the given ID.

Parameters

No parameters.

Returns

Returns the subscription object.

Update a subscription

Updates an existing subscription to match the specified parameters. When changing prices or quantities, we optionally prorate the price we charge next month to make up for any price changes. To preview how the proration is calculated, use the upcoming invoice endpoint.

By default, we prorate subscription changes. For example, if a customer signs up on May 1 for a ₹100 price, they'll be billed ₹100 immediately. If on May 15 they switch to a ₹200 price, then on June 1 they'll be billed ₹250 (₹200 for a renewal of her subscription, plus a ₹50 prorating adjustment for half of the previous month's ₹100 difference). Similarly, a downgrade generates a credit that is applied to the next invoice. We also prorate when you make quantity changes.

Switching prices does not normally change the billing date or generate an immediate charge unless:

- The billing interval is changed (for example, from monthly to yearly).
- The subscription moves from free to paid, or paid to free.
- A trial starts or ends.

In these cases, we apply a credit for the unused time on the previous price, immediately charge the customer using the new price, and reset the billing date.

If you want to charge for an upgrade immediately, pass `proration_behavior` as `always_invoice` to create prorations, automatically invoice the customer for those proration adjustments, and attempt to collect payment. If you pass `create_prorations`, the prorations are created but not automatically invoiced. If you want to bill the customer for the prorations before the subscription's renewal date, you need to manually invoice the customer or use `pending_invoice_item_interval`.

If you don't want to prorate, set the `proration_behavior` option to `none`. With this option, the customer is billed ₹100 on May 1 and ₹200 on June 1. Similarly, if you set `proration_behavior` to `none` when switching between different billing intervals (for example, from monthly to yearly), we don't generate any credits for the old subscription's unused time. We still reset the billing date and bill immediately for the new subscription.

Updating the quantity on a subscription many times in an hour may result in rate limiting. If you need to bill for a frequently changing quantity, consider integrating usage-based billing

instead.

Parameters

`cancel_at_period_end` optional
Boolean indicating whether this subscription should cancel at the end of the current period.

`default_payment_method` optional
ID of the default payment method for the subscription. It must belong to the customer associated with the subscription. This takes precedence over `default_source`. If neither are set, invoices will use the customer's invoice_settings.default_payment_method or default_source.

`description` optional
The subscription's description, meant to be displayable to the customer. Use this field to optionally store an explanation of the subscription for rendering in Stripe surfaces.

`items` optional array of hashes
A list of up to 20 subscription items, each with an attached price.

+ Show child parameters

`metadata` optional dictionary
Set of key-value pairs 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`.

`payment_behavior` optional enum
Use `allow_incomplete` to transition the subscription to `status=past_due` if a payment is required but cannot be paid. This allows you to manage scenarios where additional user actions are needed to pay a subscription's invoice. For example, SCA regulation may require 3DS authentication to complete payment. See the SCA Migration Guide for Billing to learn more. This is the default behavior.

Use `default_incomplete` to transition the subscription to `status=past_due` when payment is required and await explicit confirmation of the invoice's payment intent. This allows simpler management of scenarios where additional user actions are needed to pay a subscription's invoice. Such as failed payments, SCA regulation, or collecting a mandate for a bank debit payment method.

Use `pending_if_incomplete` to update the subscription using pending updates. When you use `pending_if_incomplete` you can only pass the parameters supported by pending updates.

Use `error_if_incomplete` if you want Stripe to return an HTTP 402 status code if a subscription's invoice cannot be paid. For example, if a payment method requires 3DS

authentication due to SCA regulation and further user action is needed, this parameter does not update the subscription and returns an error instead. This was the default behavior for API versions prior to 2019-03-14. See the changelog to learn more.

Possible enum values
`allow_incomplete`
`error_if_incomplete`
`pending_if_incomplete`
`default_incomplete`

`proration_behavior` optional enum

Determines how to handle prorations when the billing cycle changes (e.g., when switching plans, resetting `billing_cycle_anchor=now`, or starting a trial), or if an item's `quantity` changes. The default value is `create_prorations`.

Possible enum values
`create_prorations` Will cause proration invoice items to be created when applicable. These proration items will only be invoiced immediately under certain conditions.
`none` Disable creating prorations in this request.
`always_invoice` Always invoice immediately for prorations.

More parameters

Expand all

- **`add_invoice_items`** optional array of hashes
- **`application_fee_percent`** optional CONNECT ONLY
- **`automatic_tax`** optional dictionary
- **`billing_cycle_anchor`** optional
- **`billing_thresholds`** optional dictionary
- **`cancel_at`** optional

- **cancellation_details** optional dictionary
- **collection_method** optional
- **coupon** optional
- **days_until_due** optional
- **default_source** optional
- **default_tax_rates** optional
- **off_session** optional
- **on_behalf_of** optional
- **pause_collection** optional dictionary
- **payment_settings** optional dictionary
- **pending_invoice_item_interval** optional dictionary
- **promotion_code** optional
- **proration_date** optional
- **transfer_data** optional dictionary CONNECT ONLY
- **trial_end** optional
- **trial_from_plan** optional
- **trial_settings** optional dictionary

Returns

The newly updated `Subscription` object, if the call succeeded.

If `payment_behavior` is `error_if_incomplete` and a charge is required for the update and it fails, this call returns an error, and the subscription update does not go into effect.

Resume a subscription

Initiates resumption of a paused subscription, optionally resetting the billing cycle anchor and creating prorations. If a resumption invoice is generated, it must be paid or marked uncollectible before the subscription will be unpaused. If payment succeeds the subscription will become `active`, and if payment fails the subscription will be `past_due`. The resumption invoice will void automatically if not paid by the expiration date.

Parameters

`billing_cycle_anchor` optional

Either `now` or `unchanged`. Setting the value to `now` resets the subscription's billing cycle anchor to the current time (in UTC). Setting the value to `unchanged` advances the subscription's billing cycle anchor to the period that surrounds the current time. For more information, see the billing cycle documentation.

`proration_behavior` optional enum

Determines how to handle prorations when the billing cycle changes (e.g., when switching plans, resetting `billing_cycle_anchor=now`, or starting a trial), or if an item's `quantity` changes. The default value is `create_prorations`.

Possible enum values
`create_prorations` Will cause proration invoice items to be created when applicable. These proration items will only be invoiced immediately under certain conditions.
`none` Disable creating prorations in this request.
`always_invoice` Always invoice immediately for prorations.

More parameters

Expand all

`proration_date` optional

Returns

The subscription object.

Cancel a subscription

Cancels a customer's subscription immediately. The customer will not be charged again for the subscription.

Note, however, that any pending invoice items that you've created will still be charged for at the end of the period, unless manually deleted. If you've set the subscription to cancel at the end of the period, any pending prorations will also be left in place and collected at the end of the period. But if the subscription is set to cancel immediately, pending prorations will be removed.

By default, upon subscription cancellation, Stripe will stop automatic collection of all finalized invoices for the customer. This is intended to prevent unexpected payment attempts after the customer has canceled a subscription. However, you can resume automatic collection of the invoices manually after subscription cancellation to have us proceed. Or, you could check for unpaid invoices before allowing the customer to cancel the subscription at all.

Parameters

Expand all

- **`cancellation_details`** optional dictionary
- **`invoice_now`** optional
- **`prorate`** optional

Returns

The canceled `Subscription` object. Its subscription status will be set to `canceled`.

List subscriptions

By default, returns a list of subscriptions that have not been canceled. In order to list canceled subscriptions, specify `status=canceled`.

Parameters

customer optional

The ID of the customer whose subscriptions will be retrieved.

price optional

Filter for subscriptions that contain this recurring price ID.

status optional enum

The status of the subscriptions to retrieve. Passing in a value of `canceled` will return all canceled subscriptions, including those belonging to deleted customers. Pass `ended` to find subscriptions that are canceled and subscriptions that are expired due to incomplete payment. Passing in a value of `all` will return subscriptions of all statuses. If no value is supplied, all subscriptions that have not been canceled are returned.

Possible enum values
`active`
`past_due`
`unpaid`
`canceled`
`incomplete`
`incomplete_expired`
`trialing`
`paused`
`all`
`ended`

More parameters

Expand all

> **collection_method** optional

> **created** optional dictionary

> **current_period_end** optional dictionary

> **current_period_start** optional dictionary

ending_before optional

limit optional

starting_after optional

test_clock optional

Returns

Returns a list of subscriptions.

Search subscriptions

Search for subscriptions you've previously created using Stripe's Search Query Language. Don't use search in read-after-write flows where strict consistency is necessary. Under normal operating conditions, data is searchable in less than a minute. Occasionally, propagation of new or updated data can be up to an hour behind during outages. Search functionality is not available to merchants in India.

Parameters

query REQUIRED
The search query string. See search query language and the list of supported query fields for subscriptions.

limit optional
A limit on the number of objects to be returned. Limit can range between 1 and 100, and the default is 10.

page optional
A cursor for pagination across multiple pages of results. Don't include this parameter on the first call. Use the next_page value returned in a previous response to request subsequent results.

Returns

A dictionary with a `data` property that contains an array of up to `limit` subscriptions. If no objects match the query, the resulting array will be empty. See the related guide on expanding properties in lists.