
stripe-ruby-mock

- Homepage: <https://github.com/stripe-ruby-mock/stripe-ruby-mock>
- Issues: <https://github.com/stripe-ruby-mock/stripe-ruby-mock/issues>
- **CHAT**: <https://gitter.im/rebelidealist/stripe-ruby-mock>

REQUEST: Looking for More Core Contributors

This gem has unexpectedly grown in popularity and I've gotten pretty busy, so I'm currently looking for more core contributors to help me out. If you're interested, there is only one requirement: submit a significant enough pull request and have it merged into master (many of you have already done this). Afterwards, ping @gilbert in chat and I will add you as a collaborator.

Install

In your gemfile:

```
1 gem 'stripe-ruby-mock', '~> 3.1.0', :require => 'stripe_mock'
```

!!! Important

We have changelog. It's first attempt. Feel free to update it and suggest to a new format of it.

version 3.0.0 has breaking changes - we support stripe > 5 and < 6 for now and try to follow the newest API version. But if you still use older versions please read.

Features

- No stripe server access required
- Easily test against stripe errors
- Mock and customize stripe webhooks
- Flip a switch to run your tests against Stripe's **live test servers**

Requirements

- ruby >= 2.6.0
- stripe >= 5.0.0

Specifications

STRIPE API TARGET VERSION: 2019-08-20 (master) - we try, but some features are not implemented yet.

Older API version branches:

- api-2015-09-08 - use gem version 2.4.1
- api-2014-06-17

Versioning System

Since StripeMock tries to keep up with Stripe's API version, its version system is a little different:

- The **major** number (1.x.x) is for breaking changes involving how you use StripeMock itself
- The **minor** number (x.1.x) is for breaking changes involving Stripe's API
- The **patch** number (x.x.0) is for non-breaking changes/fixes involving Stripe's API, or for non-breaking changes/fixes/features for StripeMock itself.

Description

*** WARNING: This library does not cover all Stripe API endpoints. If you need one that's missing, please create an issue for it, or see this wiki page if you're interested in contributing ***

At its core, this library overrides stripe-ruby's request method to skip all http calls and instead directly return test data. This allows you to write and run tests without the need to actually hit stripe's servers.

You can use stripe-ruby-mock with any ruby testing library. Here's a quick dummy example with RSpec:

```
1 require 'stripe_mock'
2
3 describe MyApp do
4   let(:stripe_helper) { StripeMock.create_test_helper }
5   before { StripeMock.start }
6   after { StripeMock.stop }
7
8   it "creates a stripe customer" do
9
10    # This doesn't touch stripe's servers nor the internet!
11    # Specify :source in place of :card (with same value) to return
12    #   customer with source data
13    customer = Stripe::Customer.create({
```

```
13     email: 'johnny@appleseed.com',
14     source: stripe_helper.generate_card_token
15   })
16   expect(customer.email).to eq('johnny@appleseed.com')
17 end
18 end
```

Test Helpers

Some Stripe API calls require several parameters. StripeMock helps you keep your test brief with some helpers:

```
1 describe MyApp do
2   let(:stripe_helper) { StripeMock.create_test_helper }
3
4   it "creates a stripe plan" do
5     plan = stripe_helper.create_plan(:id => 'my_plan', :amount => 1500)
6
7     # The above line replaces the following:
8     # plan = Stripe::Plan.create(
9     #   :id => 'my_plan',
10    #   :name => 'StripeMock Default Plan ID',
11    #   :amount => 1500,
12    #   :currency => 'usd',
13    #   :interval => 'month'
14    # )
15    expect(plan.id).to eq('my_plan')
16    expect(plan.amount).to eq(1500)
17  end
18 end
```

The available helpers are:

```
1 stripe_helper.create_plan(my_plan_params)
2 stripe_helper.delete_plan(my_plan_params)
3 stripe_helper.generate_card_token(my_card_params)
```

For everything else, use Stripe as you normally would (i.e. use Stripe as if you were not using StripeMock).

Live Testing

Every once in a while you want to make sure your tests are actually valid. StripeMock has a switch that allows you to run your test suite (or a subset thereof) against Stripe's live test servers.

Here is an example of setting up your RSpec (2.x) test suite to run live with a command line switch:

```
1 # RSpec 2.x
2 RSpec.configure do |c|
3   if c.filter_manager.inclusions.keys.include?(:live)
4     StripeMock.toggle_live(true)
5     puts "Running **live** tests against Stripe..."
6   end
7 end
```

With this you can run live tests by running `rspec -t live`

Here is an example of setting up your RSpec (3.x) test suite to run live with the same command line switch:

```
1 # RSpec 3.x
2 RSpec.configure do |c|
3   if c.filter_manager.inclusions.rules.include?(:live)
4     StripeMock.toggle_live(true)
5     puts "Running **live** tests against Stripe..."
6   end
7 end
```

Mocking Card Errors

** Ensure you start StripeMock in a before filter `StripeMock.start` Tired of manually inputting fake credit card numbers to test against errors? Tire no more!

```
1 it "mocks a declined card error" do
2   # Prepares an error for the next create charge request
3   StripeMock.prepare_card_error(:card_declined)
4
5   expect { Stripe::Charge.create(amount: 1, currency: 'usd') }.to
6     raise_error {|e|
7       expect(e).to be_a Stripe::CardError
8       expect(e.http_status).to eq(402)
9       expect(e.code).to eq('card_declined')
10    }
11 end
```

Built-In Card Errors

```
1 StripeMock.prepare_card_error(:incorrect_number)
2 StripeMock.prepare_card_error(:invalid_number)
3 StripeMock.prepare_card_error(:invalid_expiry_month)
4 StripeMock.prepare_card_error(:invalid_expiry_year)
5 StripeMock.prepare_card_error(:invalid_cvc)
6 StripeMock.prepare_card_error(:expired_card)
```

```
7 StripeMock.prepare_card_error(:incorrect_cvc)
8 StripeMock.prepare_card_error(:card_declined)
9 StripeMock.prepare_card_error(:missing)
10 StripeMock.prepare_card_error(:processing_error)
11 StripeMock.prepare_card_error(:incorrect_zip)
```

You can see the details of each error in `lib/stripe_mock/api/errors.rb`

Specifying Card Errors

** Ensure you start StripeMock in a before filter `StripeMock.start` By default, `prepare_card_error` only triggers for `:new_charge`, the event that happens when you run `Charge.create`. More explicitly, this is what happens by default:

```
1 StripeMock.prepare_card_error(:card_declined, :new_charge)
```

If you want the error to trigger on a different event, you need to replace `:new_charge` with a different event. For example:

```
1 StripeMock.prepare_card_error(:card_declined, :create_card)
2 customer = Stripe::Customer.create
3 # This line throws the card error
4 customer.cards.create
```

`:new_charge` and `:create_card` are names of methods in the StripeMock request handlers. You can also set `StripeMock.toggle_debug(true)` to see the event name for each Stripe request made in your tests.

Custom Errors

** Ensure you start StripeMock in a before filter `StripeMock.start` To raise an error on a specific type of request, take a look at the request handlers folder and pass a method name to `StripeMock.prepare_error`.

If you wanted to raise an error for creating a new customer, for instance, you would do the following:

```
1 it "raises a custom error for specific actions" do
2   custom_error = StandardError.new("Please knock first.")
3
4   StripeMock.prepare_error(custom_error, :new_customer)
5
6   expect { Stripe::Charge.create(amount: 1, currency: 'usd') }.to_not
7     raise_error
8   expect { Stripe::Customer.create }.to raise_error { |e|
9     expect(e).to be_a StandardError
10  }
```

```
9     expect(e.message).to eq("Please knock first.")
10   }
11 end
```

In the above example, `:new_customer` is the name of a method from `customers.rb`.

Running the Mock Server

Sometimes you want your test stripe data to persist for a bit, such as during integration tests running on different processes. In such cases you'll want to start the stripe mock server:

```
1 # spec_helper.rb
2 #
3 # The mock server will automatically be killed when your tests are done
  running.
4 #
5 require 'thin'
6 StripeMock.spawn_server
```

Then, instead of `StripeMock.start`, you'll want to use `StripeMock.start_client`:

```
1 describe MyApp do
2   before do
3     @client = StripeMock.start_client
4   end
5
6   after do
7     StripeMock.stop_client
8     # Alternatively:
9     # @client.close!
10    # -- Or --
11    # StripeMock.stop_client(:clear_server_data => true)
12  end
13 end
```

This is all essentially the same as using `StripeMock.start`, except that the stripe test data is held in its own server process.

Here are some other neat things you can do with the client:

```
1 @client.state ==> 'ready'
2
3 @client.get_server_data(:customers) # Also works for :charges, :plans,
  etc.
4 @client.clear_server_data
5
6 @client.close!
7 @client.state ==> 'closed'
```

Mock Server Options

```
1 # NOTE: Shown below are the default options
2 StripeMock.default_server_pid_path = './stripe-mock-server.pid'
3
4 StripeMock.spawn_server(
5   :pid_path => StripeMock.default_server_pid_path,
6   :host => '0.0.0.0',
7   :port => 4999,
8   :server => :thin
9 )
10
11 StripeMock.kill_server(StripeMock.default_server_pid_path)
```

Mock Server Command

If you need the mock server to continue running even after your tests are done, you'll want to use the executable:

```
1 $ stripe-mock-server -p 4000
2 $ stripe-mock-server --help
```

Mocking Webhooks

If your application handles stripe webhooks, you are most likely retrieving the event from stripe and passing the result to a handler. StripeMock helps you by easily mocking that event:

```
1 it "mocks a stripe webhook" do
2   event = StripeMock.mock_webhook_event('customer.created')
3
4   customer_object = event.data.object
5   expect(customer_object.id).to_not be_nil
6   expect(customer_object.default_card).to_not be_nil
7   # etc.
8 end
9
10 it "mocks stripe connect webhooks" do
11   event = StripeMock.mock_webhook_event('customer.created', account: '
12     acc_123123')
13   expect(event.account).to eq('acc_123123')
14 end
```

Customizing Webhooks

By default, StripeMock searches in your `spec/fixtures/stripe_webhooks/` folder for your own, custom webhooks. If it finds nothing, it falls back to test events generated through stripe's webhooktester.

For example, you could create a file in `spec/fixtures/stripe_webhooks/invoice.created.with-sub.json`, copy/paste the default from the default `invoice.created.json`, and customize it to your needs.

Then you can use that webhook directly in your specs:

```
1 it "can use a custom webhook fixture" do
2   event = StripeMock.mock_webhook_event('invoice.created.with-sub')
3   # etc.
4 end
```

You can also override values on the fly:

```
1 it "can override webhook values" do
2   # NOTE: given hash values get merged directly into event.data.object
3   event = StripeMock.mock_webhook_event('customer.created', {
4     :id => 'cus_my_custom_value',
5     :email => 'joe@example.com'
6   })
7   # Alternatively:
8   # event.data.object.id = 'cus_my_custom_value'
9   # event.data.object.email = 'joe@example.com'
10  expect(event.data.object.id).to eq('cus_my_custom_value')
11  expect(event.data.object.email).to eq('joe@example.com')
12 end
```

You can name events whatever you like in your `spec/fixtures/stripe_webhooks/` folder. However, if you try to call a non-standard event that's doesn't exist in that folder, StripeMock will throw an error.

If you wish to use a different fixture path, you can set it yourself:

```
1 StripeMock.webhook_fixture_path = './spec/other/folder/'
```

Generating Card Tokens

Sometimes you need to check if your code reads a stripe card correctly. If so, you can specifically assign card data to a generated card token:

```
1 it "generates a stripe card token" do
```

```
2   card_token = StripeMock.generate_card_token(last4: "9191", exp_year:
    1984)
3
4   cus = Stripe::Customer.create(source: card_token)
5   card = cus.sources.data.first
6   expect(card.last4).to eq("9191")
7   expect(card.exp_year).to eq(1984)
8   end
```

Debugging

To enable debug messages:

```
1 StripeMock.toggle_debug(true)
```

This will **only last for the session**; Once you call `StripeMock.stop` or `StripeMock.stop_client`, debug will be toggled off.

If you always want debug to be on (it's quite verbose), you should put this in a `before` block.

Miscellaneous Features

You may have noticed that all generated Stripe ids start with `test_`. If you want to remove this:

```
1 # Turns off test_ prefix
2 StripeMock.global_id_prefix = false
3
4 # Or you can set your own
5 StripeMock.global_id_prefix = 'my_app_'
```

TODO

- Cover all stripe urls/methods
- Throw useful errors that emulate Stripe's requirements
 - For example: "You must supply either a card or a customer id" for `Stripe::Charge`
- Fingerprinting for other resources besides Cards

Developing stripe-ruby-mock

Please see this wiki page

Patches are welcome and greatly appreciated! If you're contributing to fix a problem, be sure to write tests that illustrate the problem being fixed. This will help ensure that the problem remains fixed in future updates.

Note: You may need to `ulimit -n 4048` before running the test suite to get all tests to pass.

Copyright

Copyright (c) 2013 Gilbert

See LICENSE.txt for details.