

# Integrating the PayPal API with Ruby on Rails

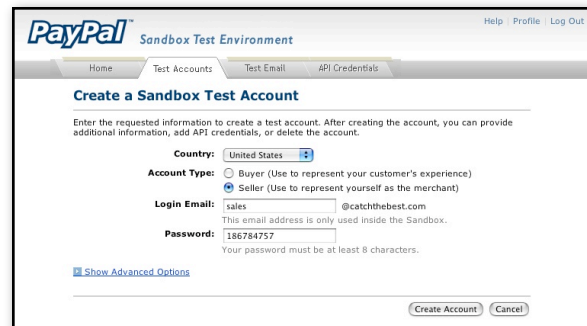
This guide will help you integrate the PayPal Website Payments Pro API into your Rails application in 20 minutes or less. Included with this book is a fully-functional sample Rails application that has all the code you need to see how to integrate PayPal into your own application. Let's get started!

## Playing in the Sandbox

The first thing you need to do is set up your developer account with PayPal by visiting the [PayPal Developer Central](#) web site.

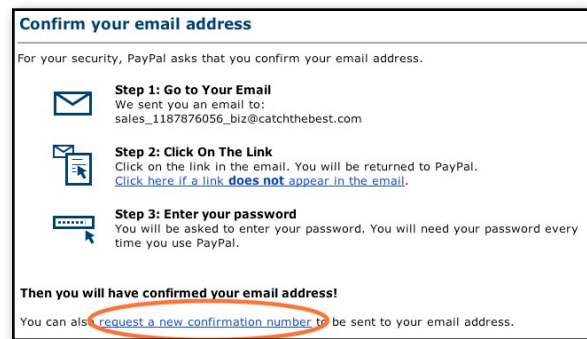
You'll need to enter some basic contact info and a valid email address to create your developer account. Once you've done that and verified your account by clicking on the link in the account verification email, you can log in to your developer account and start creating Sandbox accounts.

Click the Sandbox tab and then the Create Test Account link to get started. The first Sandbox account you'll need to create is the seller account. Note the password that you select when creating the seller account, as you'll be using that to log in as the seller in the Sandbox environment in just a minute. Go back to the Test Accounts tab and click the Disabled link next to the account you just created to enable the account.

The screenshot shows the 'PayPal Sandbox Test Environment' interface. At the top, there's a navigation bar with 'Home', 'Test Accounts', 'Test Email', and 'API Credentials'. The 'Test Accounts' tab is active. Below the navigation bar, the heading is 'Create a Sandbox Test Account'. A sub-header says 'Enter the requested information to create a test account. After creating the account, you can provide additional information, add API credentials, or delete the account.' The form fields include: 'Country' (dropdown menu set to 'United States'), 'Account Type' (radio buttons for 'Buyer (Use to represent your customer's experience)' and 'Seller (Use to represent yourself as the merchant)', 'Login Email' (text input with 'sales' and '@catchthebest.com'), and 'Password' (text input with '186784757'). A note below the password field states 'This email address is only used inside the Sandbox.' and 'Your password must be at least 8 characters.' There is a 'Show Advanced Options' link and 'Create Account' and 'Cancel' buttons at the bottom right.

The Sandbox environment can be a bit flaky, so you may have to do a few extra steps to get your account to work. For example, if you click the Show Advanced Options link on the account creation form, you'll see that it defaults to automatically confirming the bank account information and the email address for the new account. While writing this book, I found I always got an error when creating an account that had the Confirm Email option turned on. When going back to the list, it would show the new account, and I could launch the Sandbox for that account, but the email address would actually not be confirmed, and I couldn't get PayPal to actually sent the account confirmation email (remember, all email is delivered to the Test Email tab of the Sandbox site). I found that if I created the test account without the email confirmed, then I could get the confirmation email sent,

but only if I clicked on the “request a new confirmation number” link after clicking on the “confirm” button from the main account page. I also found that I couldn’t get the account confirmed by following the link in the account confirmation email -- I had to choose the “click here if a link does not appear in the email” link and enter the confirmation number that was provided in the email. Hopefully you won’t have the same problems I had, but if you do, then hopefully you’ll get spared some head-scratching as to why things aren’t working.



After you’ve created the seller account, log in to the test seller account from the Test Account list. You’ll need to complete the step of confirming the account’s email address as just mentioned. Then you can configure your seller account for Website Payments Pro by completing the application process. If you’re lucky, the Sandbox created your seller test account with an application already complete for the Website Payments Pro service, and you’ll see that you are ready for step two of the payment solution setup: accepting the billing agreement. If you’re unlucky, you’ll have to complete a Website Payments Pro application, which includes providing info about your business, and guessing a social security number starting with 111 that hasn’t been used by anyone else. Once you accept the billing agreement (you won’t really be charged \$20 / month for your test account), view the API credentials for the test account. Note the API username, password, and signature for later.



While you’re in the sandbox, go ahead and create a buyer account as well. Save the credit card info generated for the buyer in a note somewhere so you can use it to do some testing later.

## The ActiveMerchant Plugin

Thanks to the brilliant guys at [Shopify](#), the Rails world gets to enjoy the benefits of the [ActiveMerchant plugin](#). This plugin abstracts away most of the differences among the various payment processors, giving you a consistent interface for taking people’s money. The sample application already has the plugin installed.

To configure the application and the ActiveMerchant plugin for your PayPal login info, edit `config/paypal.yml` and enter the username, password, and signature you got from the PayPal Sandbox in the development section. You would put your live account credentials in the production section.

## The Controller

Alright, it's time to get into some Rails code. Follow along by opening up the sample application. Before you dig in too deep, take a little side trip over to `config/routes.rb` and notice the default route. I have connected the default route to `PurchaseController` (and removed `public/index.html`) to make it super-speedy for you to start up the Rails app and started running transactions against the PayPal test environment. You can thank me later.

Ok, now it's **really** time to get into some rails code. Open `app/controllers/purchase_controller.rb` to get down to business. You'll see that I have a constant at the top of the controller that sets the amount to charge at \$12. I'm sure you'll come up with a more creative way to determine the price for your product or service, but I needed something simple for the example.

After that you can see the simple index action that just renders the view with the credit card form and the link to make a payment at PayPal. Let me digress a minute here and remind you of one of the requirements of using Website Payments Pro as your processor for credit card transactions: Express Checkout. PayPal requires that you support the ability for customers to use Express Checkout (going to PayPal to enter payment information) as well as the ability for customers to enter credit card information at your site. The sample application has support for both methods.

### Capturing and processing credit card (Direct Payment) transactions

The credit action receives the POST from the credit card form. The first step is to validate the credit card instance variable, which is loaded by the before filter. The validation provided by the `ActiveMerchant::CreditCard` class includes things like checking the validity of the number (via the [Luhn algorithm](#)), that the expiration date is in the future, etc. Once the validity is checked, an instance of `ActiveMerchant::Paypal` gateway is used to send a purchase transaction to the PayPal servers.

The `paypal_gateway` method, which returns an `ActiveMerchant` gateway instance, is located at the bottom of the controller (line 74). In that method, you can see that the login credentials defined in `config/environments/development.rb` are used when creating the gateway object. This object handles generating the XML for the request, sending the XML to PayPal's servers, and receiving and parsing the response from PayPal.

If the gateway transaction was successful (line 21), then a `Purchase` record is created with the info from the PayPal response. I'll get to details about the model a little later. After creating the `Purchase` record, the user is redirected to a "thank you" page with the ID of the newly created record so that details of the record can be retrieved and displayed on that page.

### Handling Express Checkout transactions

The express action is invoked when a customer clicks on the "Pay at PayPal" icon on the index page. Line 50 creates a `PaypalExpress` gateway, rather than the `Paypal` gateway used previously (again by using the method at line 74), and then invokes the `setup_purchase` method to let PayPal know about the transaction to be processed and to get a token to be

used for sending the customer to the PayPal payment page with the correct transaction information. If the setup of the transaction was successful (line 39), the customer is then redirected to PayPal to enter the payment info and complete the transaction. It is at this point that you could create a record in the database (e.g., a pending order) if you wanted to keep track of customers that never return from PayPal.

Customers returning from PayPal after entering the payment information get sent to the `express_complete` action (because of the `return_url` option to the `setup_purchase` method on line 34). This action again creates a `PaypalExpress` gateway and uses it to check the status of the transaction that was just completed. This is done by calling the `details_for` method of the gateway with a token that came back with the customer when being redirected to the `express_complete` action. If the token was valid and there are completion details for the PayPal transaction (line 53), then the gateway's `purchase` method is invoked to actually complete the charge for the customer. If that executed successfully, then a `Purchase` record is created with the information from the PayPal transaction, and the customer is redirected to the "thank you" page.

### **A quick word on security**

Before I move from the controller to the model, there are a couple of things you should notice in the controller regarding the security of your customer's credit card data.

You'll notice at the top of the controller that I have included the code from the [ssl\\_requirement plugin](#). This plugin helps ensure that certain actions are executed only via SSL, which of course is a requirement for receiving credit card information. I have actually modified the plugin (line 50 of `ssl_requirement.rb`) to not redirect to SSL if the app isn't running in the production environment. This is simply so you can try out the app without having to worry about setting up SSL.

On line 7 of the controller you'll see that the credit card data submitted from the form is prevented from being logged. Just as you must keep the customer's data secure by using SSL, you must also keep the customer's data secure by not logging it to a text file on your server.

## **The Model**

The model is fairly simple. The first thing you'll notice is the use of the `TokenGenerator` module (in `lib/`). This plugin just adds an globally unique ID (GUID) to your model when saving it, so you can have another unique identifier besides the primary key. Having this token allows us to have something in the URL or something the customer can use to refer to an order without revealing the ID of the record in the database.

The `to_param` method is used to override the usual ID in the URL when the purchase is referenced in a route. This comes into play on lines 23 and 59 of the controller when redirecting to display the purchase record after the transaction has been completed. Rather than exposing the ID, `to_param` exposes the generated token.

The `response=` method sets a number of model attributes based on the response that is returned from the PayPal gateway object. Lines 12-14 set the attributes based on the info received from a credit card transaction, while line 16 handles the special case of the purchase amount being returned in the “`gross_amount`” parameter from the Express Checkout response.

## Wrapping Up

I hope this guide was useful to you and saved you some time in adding PayPal to your Rails application. If you have any questions about using the ActiveMerchant plugin, please check out the [ActiveMerchant Google Group](#). If you have any questions or comments about the material in this guide, please send them to me at [ben@bencurtis.com](mailto:ben@bencurtis.com). If you'd like to read more about using Rails to build e-commerce applications, do take a look at my e-book [The Money Train](#), which provides a number of helpful hints for building a shopping cart with Rails.