

Ruby(<https://www.sitepoint.com/ruby/>) - September 01, 2016 - By Ilya Bodrov-Krukowski (<https://www.sitepoint.com/author/ibodrov/>)

## Better Nested Attributes in Rails with the Cocoon Gem

In this article we are going to discuss how to build more complex forms using Rails' nested attributes feature. I will show you how to manipulate multiple associated records from a single form and properly set up the models and controller to enable this feature. Also, we are going to discuss common pitfalls and power our form up to make it more dynamic using the gem called Cocoon (<https://github.com/nathanvda/cocoon>). This solution allows adding and removing nested fields asynchronously while providing lots of customization options and callbacks.

The source code is available at GitHub (<https://github.com/bodrovis/Sitepoint-source/tree/master/NestedForms>).

The demo app is available at sitepoint-nested-forms.herokuapp.com (<https://sitepoint-nested-forms.herokuapp.com/>).

## Building a Simple Form

*For this demo I'll be using Rails 5 but most of the described concepts can be applied to Rails 3 and 4*

Go ahead and create a new application without the default testing suite:

```
$ rails new NestedForms -T
```

Suppose that, with this app, we want to keep track of our favorite places and their addresses. For example, if we enter "Cafe" as a place along with a bunch of addresses of our preferred cafes. This means that one place may have many addresses, so we'll describe it using associations:

```
$ rails g model Place title:string
$ rails g model Address city:string street:string place:belongs_to
$ rake db:migrate
```

Make sure that the associations are set up properly:

*models/place.rb*

```
[...]
has_many :addresses, dependent: :destroy
[...]
```

*models/address.rb*

```
[...]
belongs_to :place
[...]
```

Now code a basic **PlacesController** (the one to rule them all...):

*app/controllers/places\_controller.rb*

```

class PlacesController < ApplicationController
  def index
    @places = Place.all
  end

  def new
    @place = Place.new
  end

  def create
    @place = Place.new(place_params)
    if @place.save
      redirect_to root_path
    else
      render :new
    end
  end

  private

  def place_params
    params.require(:place).permit(:title)
  end
end

```

Add the routes:

*config/routes.rb*

```

[...]
resources :places, only: [:new, :create, :edit, :update]

root to: 'places#index'
[...]

```

Now, the view for the root page:

*views/places/index.html.erb*

```

<h1>Places</h1>

<p><%= link_to 'Add place', new_place_path %></p>

<ul><%= render @places %></ul>

```

Having added **render @places**, we also need the corresponding partial:

*views/places/\_place.html.erb*

```

    <li>
      <strong><%= place.title %></strong><br>
      <% if place.addresses.any? %>
        Addresses:
        <ul>
          <% place.addresses.each do |addr| %>
            <li>
              <%= addr.city %>, <%= addr.street %>
            </li>
          <% end %>
        </ul>
      <% end %>
    </li>

```

The view to create places:

*views/places/new.html.erb*

```

<h1>Add place</h1>

<%= render 'form' %>

```

Including the form:

*views/places/\_form.html.erb*

```

<%= render 'shared/errors', object: @place %>

<%= form_for @place do |f| %>
  <div>
    <%= f.label :title %>
    <%= f.text_field :title %>
  </div>

  <%= f.submit %>
<% end %>

```

Here's yet another partial to display errors:

*views/shared/\_errors.html.erb*

```

<% if object.errors.any? %>
  <div>
    <strong>
      <%= pluralize(object.errors.count, 'error') %> were found
    </strong>

    <ul>
      <% object.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>

```

So far so good. However, in terms of better user experience, I'd like to allow adding place's addresses on the same page rather than coding a separate form. This will also save us from coding an additional controller to manipulate addresses. That's where the nested attributes come into play.

# Adding Nested Attributes

The idea behind the nested attributes ([http://guides.rubyonrails.org/form\\_helpers.html#building-complex-forms](http://guides.rubyonrails.org/form_helpers.html#building-complex-forms)) is quite simple. You have a single form where you can create an object along with its associated records. This feature can be added really fast, as it requires very small modifications to the controller and the model, as well as some markup.

It all starts with the addition of the long-named accepts\_nested\_attributes\_for ([http://apidock.com/rails/ActiveRecord/NestedAttributes/ClassMethods/accepts\\_nested\\_attributes\\_for](http://apidock.com/rails/ActiveRecord/NestedAttributes/ClassMethods/accepts_nested_attributes_for)) method:

*models/places.rb*

```
[...]
accepts_nested_attributes_for :addresses
[...]
```

Having added this method, we can now manipulate addresses via the places' mass-assignment mechanism. The controller requires some changes as well:

*places\_controller.rb*

```
[...]
private

def place_params
  params.require(:place).permit(:title, addresses_attributes: [:id, :city, :street])
end
[...]
```

When you submit a form with the nested fields, the `params[:place]` will contain an array under the key `:addresses_attributes`. This array describes each address to be added into the database. As long as we are using `strong_params`, those new attributes have to be explicitly permitted.

Now add the nested form into the view:

*views/places/\_form.html.erb*

```
<%= form_for @place do |f| %>
  <%= render 'shared/errors', object: @place %>
  <div>
    <%= f.label :title %>
    <%= f.text_field :title %>
  </div>

  <div>
    <p><strong>Addresses:</strong></p>

    <%= f.fields_for :addresses do |address| %>
      <div>
        <%= address.label :city %>
        <%= address.text_field :city %>

        <%= address.label :street %>
        <%= address.text_field :street %>
      </div>
    <% end %>
  </div>

  <%= f.submit %>
<% end %>
```

The `fields_for` ([http://api.rubyonrails.org/classes/ActionView/Helpers/FormBuilder.html#method-i-fields\\_for](http://api.rubyonrails.org/classes/ActionView/Helpers/FormBuilder.html#method-i-fields_for)) method, as you've probably guessed, adds the nested fields. It is quite similar to the `form_for` method but does not provide the `form` tag itself. Note that inside the block I am using a new local variable `address` – *do not* call it `f` because it already contains the builder for the parent form.

There is a problem, however. When you visit the “New Place” page you won’t see any nested fields, because obviously the new instance of the `Place` class does not contain any nested addresses. The simple fix, as suggested by the Rails docs, would be to build a couple of addresses directly in the controller:

*places\_controller.rb*

```
[...]
def new
  @place = Place.new
  3.times { @place.addresses.build}
end
[...]
```

Indeed that’s not the best solution and we’ll get rid of it later.

You may now boot the server, navigate to the “New Place” page, and try creating a place with some nested addresses. However, things can’t always go that smooth, right? If you are using Rails 5.0, like me, you’ll see a pretty strange error “Addresses place must exist” preventing the form from being submitted. This appears to be a [major bug](https://github.com/rails/rails/issues/25198) (<https://github.com/rails/rails/issues/25198>) in Rails 5 that is related to the new `belongs_to_required_by_default` option set to `true`. This setting means that the associated record must be present by default. To globally opt-out from this behaviour you may either set `Rails.application.config.active_record.belongs_to_required_by_default` to `false` (inside the *new\_framework\_defaults.rb* initializer file) or provide the `optional: true` option to the `belongs_to` method.

Another fix suggested [here \(https://github.com/rails/rails/issues/25198#issuecomment-223352645\)](https://github.com/rails/rails/issues/25198#issuecomment-223352645) involves using the `inverse_of` option:

*models/place.rb*

```
[...]
has_many :addresses, dependent: :destroy, inverse_of: :place
[...]
```

This bug should be fixed in Rails 5.1.

## A Bit of Validation

Currently, a user may create a place with a list of empty addresses, which is probably not what you want. To control this behavior, use the `reject_if` option that accepts either a lambda or the `:all_blank` value. `:all_blank` will reject a record where all the attributes are blank. However, in our case, we want to reject if *any* attribute is blank, so let’s stick with the lambda:

*models/place.rb*

```
[...]
accepts_nested_attributes_for :addresses,
                             reject_if: ->(attrs) { attrs['city'].blank? || attrs['street'].blank? }
[...]
```

Now any address without a city or street won’t be saved into the database.

# Destroy 'em

The addresses can now be added, but there is no way to remove them later. To tackle this issue, supply yet another option to the `accepts_nested_attributes_for` method:

*models/place.rb*

```
[...]
accepts_nested_attributes_for :addresses, allow_destroy: true,
                                reject_if: ->(attrs) { attrs['city'].blank? || attrs['street'].blank? }
[...]
```

This simply means that now nested records can be destroyed. In order to destroy a nested record, the `_destroy` field has to be set with a truthy value (that is 1, '1', true, or 'true'). This new field has to be permitted as well:

*places\_controller.rb*

```
[...]
private

def place_params
  params.require(:place).permit(:title, addresses_attributes: [:id, :city, :street, :_destroy])
end
[...]
```

Add a checkbox to mark the nested records for deletion:

*views/places/\_form.html.erb*

```
[...]
<div>
  <p><strong>Addresses:</strong></p>

  <%= f.fields_for :addresses do |address| %>
    <div>
      <%= address.label :city %>
      <%= address.text_field :city %>

      <%= address.label :street %>
      <%= address.text_field :street %>

      <%= address.check_box :_destroy %>
    </div>
  <% end %>
</div>
[...]
```

Now code two new controller's actions:

*places\_controller.rb*

```

    (/\
[...]
```

```

def edit
  @place = Place.find_by(id: params[:id])
end

def update
  @place = Place.find_by(id: params[:id])
  if @place.update_attributes(place_params)
    redirect_to root_path
  else
    render :edit
  end
end
[...]
```

Note that the actions themselves do not require any special changes, which is really great.

Add two more routes:

*config/routes.rb*

```

[...]
```

```

resources :places, only: [:new, :create, :edit, :update]
```

```

[...]
```

And present the “Edit” link:

*views/places/\_place.html.erb*

```

<li>
  <strong><%= place.title %></strong> | <%= link_to 'Edit place', edit_place_path(place) %><br>
  [...]
</li>
```

Now open any existing place, set the checkboxes near the addresses you wish to destroy and submit the form!

## Making It Dynamic

The basic nested form is complete, however, it is not very convenient to use. For example, there is no way to add more than three addresses. Adding this feature requires more work because Rails does not support adding fields dynamically out of the box. Luckily for us, there is already a solution available. It is called **Cocoon** (<https://github.com/nathanvda/cocoon>) and it is awesome. Cocoon powers nested forms with JavaScript, allowing files to be added or removed dynamically. Cocoon provides other customizations, as well. What’s more, it is form builder-agnostic and, therefore, works with either the basic Rails form builder or solutions like SimpleForm or Formtastic.

Getting started with Cocoon is simple. Add a new gem:

*Gemfile*

```

[...]
```

```

gem "cocoon"
```

```

[...]
```

And install it:

```
$ bundle install
```

Next, hook up a new JavaScript file:

javascripts/application.js

```
[...]
//= require cocoon
[...]
```

Note that Cocoon requires jQuery to be present. Now extract the nested fields into a separate partial:

*views/places/\_address\_fields.html.erb*

```
<div class="nested-fields">
  <%= f.label :city %>
  <%= f.text_field :city %>

  <%= f.label :street %>
  <%= f.text_field :street %>

  <%= f.check_box :_destroy %>

  <%= link_to_remove_association "remove address", f %>
</div>
```

Here we meet the first Cocoon's helper – [link to remove association](https://github.com/nathanvda/cocoon#link_to_remove_association) ([https://github.com/nathanvda/cocoon#link\\_to\\_remove\\_association](https://github.com/nathanvda/cocoon#link_to_remove_association)). This helper, as the name implies, create a new link that asynchronously deletes an associated record. This method accepts three arguments (the third one is optional):

The text to show in the link

The form object

HTML options (similar to the ones passed to the `link_to`)

Note that the `nested-fields` class is required for the “remove address” link to work.

Now we need to use this partial inside the form:

*views/places/\_form.html.erb*



```

  (f)
  <%= form_for @place do |f| %>
    <%= render 'shared/errors', object: @place %>

    <div>
      <%= f.label :title %>
      <%= f.text_field :title %>
    </div>

    <div>
      <p><strong>Addresses:</strong></p>

      <div id="addresses">
        <%= f.fields_for :addresses do |address| %>
          <%= render 'address_fields', f: address %>
        <% end %>

        <div class="links">
          <%= link_to_add_association 'add address', f, :addresses %>
        </div>
      </div>
    </div>

    <%= f.submit %>
  <% end %>

```

Here we are using the second Cocoon's helper – [link to add association](https://github.com/nathanvda/cocoon#link_to_add_association) ([https://github.com/nathanvda/cocoon#link\\_to\\_add\\_association](https://github.com/nathanvda/cocoon#link_to_add_association)). It renders a link to dynamically add nested fields using the partial we've coded a minute ago. This method accepts four parameters (the fourth one is optional):

The text to show in the link

The form builder (the parent's form, not the nested's one!)

The name of the association

HTML options. These options are similar to the ones the `link_to` accepts, however there are some special parameters available (like where to render the nested fields or which partial to use), so be sure to browse the [docs](https://github.com/nathanvda/cocoon#link_to_add_association) ([https://github.com/nathanvda/cocoon#link\\_to\\_add\\_association](https://github.com/nathanvda/cocoon#link_to_add_association))

That's pretty much it! Boot the server and try adding and removing places' addresses. This is much convenient now, isn't it?

## Cocoon's Callbacks

The last thing I am going to show you today is how to set up Cocoon's [callbacks](https://github.com/nathanvda/cocoon#callbacks-upon-insert-and-remove-of-items) (<https://github.com/nathanvda/cocoon#callbacks-upon-insert-and-remove-of-items>). There are four of them:

```

cocoon:before-insert
cocoon:after-insert
cocoon:before-remove
cocoon:after-remove

```

With the `cocoon:before-insert` you may animate the nested fields' appearance. Let's code this in a new CoffeeScript file:

*javascripts/global.coffee*

```

jQuery(document).on 'turbolinks:load', ->
  addresses = $('#addresses')

  addresses.on 'cocoon:before-insert', (e, el_to_add) ->
    el_to_add.fadeIn(1000)

```

As long as I am using Turbolinks 5 (<https://github.com/turbolinks/turbolinks-classic>), we are listening to the `turbolinks:load` event. If you prefer to stay away from Turbolinks for some reason, the first line will be much simpler:

*javascripts/global.coffee*

```
jQuery ->
```

Require this file:

*javascripts/application.js*

```
[...]
//= require global
[...]
```

Inside the `cocoon:after-insert` callback you may, for example, highlight the added fields. The jQueryUI library has a bunch of effects (<https://jqueryui.com/effect/>) to pick from – I am going to utilize the “Highlight” effect in this demo.

Add the new gem:

*Gemfile*

```
gem 'jquery-ui-rails'
```

Install it:

```
$ bundle install
```

Require a new JS file (note the proper order):

*javascripts/application.js*

```
//= require jquery
//= require jquery_ujs
//= require jquery-ui/effect-highlight
//= require cocoon
//= require global
//= require turbolinks
```

Now utilize this new effect:

*javascripts/global.coffee*

```
addresses.on 'cocoon:after-insert', (e, added_el) ->
  added_el.effect('highlight', {}, 500)
```

To animate an element’s removal, use the `cocoon:before-remove` callback. There is a small gotcha here, however. The actual removal of the element from the page has to be delayed because otherwise, we won’t be able to animate it.

*javascripts/global.coffee*

```
addresses.on 'cocoon:before-remove', (e, el_to_remove) ->
  $(this).data('remove-timeout', 1000)
  el_to_remove.fadeOut(1000)
```

`$(this).data('remove-timeout', 1000)` says Cocoon to delay the element's removal by 1 second – just enough for us to perform the animation.

Lastly, let's display how many nested records were added and change that count dynamically. Add a new `.count` block:

*views/places/\_form.html.erb*

```
[...]
<div>
  <p><strong>Addresses:</strong></p>

  <div id="addresses">
    <%= f.fields_for :addresses do |address| %>
      <%= render 'address_fields', f: address %>
    <% end %>

    <div class="links">
      <%= link_to_add_association 'add address', f, :addresses %>
    </div>

    <p class="count">Total: <span><%= @place.addresses.count %></span></p>
  </div>
</div>
[...]
```

Next, write a simple `recount` function that is going to change the counter:

*javascripts/global.coffee*

```
jQuery(document).on 'turbolinks:load', ->
  addresses = $('#addresses')
  count = addresses.find('.count > span')

  recount = -> count.text addresses.find('.nested-fields').size()
  [...]
```

Lastly, update the `cocoon:after-insert` callback and add a new one called `cocoon:after-remove`. The final version of the script is presented below:

*javascripts/global.coffee*

```
jQuery(document).on 'turbolinks:load', ->
  addresses = $('#addresses')
  count = addresses.find('.count > span')

  recount = -> count.text addresses.find('.nested-fields').size()

  addresses.on 'cocoon:before-insert', (e, el_to_add) ->
    el_to_add.fadeIn(1000)

  addresses.on 'cocoon:after-insert', (e, added_el) ->
    added_el.effect('highlight', {}, 500)
    recount()

  addresses.on 'cocoon:before-remove', (e, el_to_remove) ->
    $(this).data('remove-timeout', 1000)
    el_to_remove.fadeOut(1000)

  addresses.on 'cocoon:after-remove', (e, removed_el) ->
    recount()
```

# Limit?

You may wonder whether it is possible to limit the number of nested records somehow. The [accepts\\_nested\\_attributes\\_for](http://api.rubyonrails.org/classes/ActiveRecord/NestedAttributes/ClassMethods.html#method-i-accepts_nested_attributes_for) ([http://api.rubyonrails.org/classes/ActiveRecord/NestedAttributes/ClassMethods.html#method-i-accepts\\_nested\\_attributes\\_for](http://api.rubyonrails.org/classes/ActiveRecord/NestedAttributes/ClassMethods.html#method-i-accepts_nested_attributes_for)) method does support the `:limit` which specifies the maximum number of associated records that can be processed. It can be supplied with an integer, a procedure, or a symbol pointing to a method (both the procedure and the method must return an integer).

Cocoon, however, does not support limiting of the nested records at the time of writing this article. There was an discussion regarding this issue but the author [does not consider it](https://github.com/nathanvda/cocoon/issues/176#issuecomment-27851818) (<https://github.com/nathanvda/cocoon/issues/176#issuecomment-27851818>) to be a core feature. Still, there is [an open pull request](https://github.com/nathanvda/cocoon/pull/206) (<https://github.com/nathanvda/cocoon/pull/206>) adding this functionality available that may be merged some time in future.

## Conclusion

In this article we've discussed the usage of the nested attributes in Rails. We've created a basic, nested form allowing users to add, edit, and destroy the associated records. Later we integrated the Cocoon gem and powered our form with jQuery, making it dynamic.

Cocoon has many more options available for customization, so be sure to browse its [docs](https://github.com/nathanvda/cocoon) (<https://github.com/nathanvda/cocoon>). Hopefully, this article was useful to you. As always, I thank you for staying with me and see you soon!



Meet the author  
[\(http://www.sitepoint.com/author/ibodrov/\)](#) [\(https://twitter.com/bodrovis\)](#) [\(https://plus.google.com/103641984440210150447\)](#) [\(https://facebook.com/isbodrov\)](#) [\(https://linkedin.com/in/bodrovis\)](#) [\(https://github.com/bodrovis\)](#)








Ilya Bodrov is personal IT teacher, a senior engineer working at Campaigner LLC, author and teaching assistant at Sitepoint and lecturer at Moscow Aviations Institute. His primary programming languages are Ruby (with Rails) and JavaScript. He enjoys coding, teaching people and learning new things. Ilya also has some Cisco and Microsoft certificates and was working as a tutor in an educational center for a couple of years. In his free time he tweets, writes posts for his [website](http://bodrovis.tech) (<http://bodrovis.tech>), participates in OpenSource projects, goes in for sports and plays music.

## Login or Create Account to Comment

[Login \(/Premium/Sign-In?Ref\\_source=Sitepoint&Ref\\_medium=Comments&Redirect\\_path=%2Fbetter-Nested-Attributes-In-Rails-With-The-Cocoon-Gem%2F%23commentsSection\)](#)

[Create Account \(/Premium/Sign-Up?Ref\\_source=Sitepoint&Ref\\_medium=Comments&Redirect\\_path=%2Fbetter-Nested-Attributes-In-Rails-With-The-Cocoon-Gem%2F%23commentsSection\)](#)

## Popular In the Community

<div>HOW TO BUILD A REACT APP THAT WORKS WITH...</div> <div> Hugo Hyz 6d</div> <div>Awesome content! I'm looking to transfert...</div>	<div>HOW TO BUILD A SIMPLE BLOG USING REACT,...</div> <div> Maxime Soulie 1d</div> <div>Hello,Just after editing the package.json file,...</div>	<div>USING PREACT AS A REACT ALTERNATIVE —...</div> <div> Phil Mander 1d</div> <div>Thanks for the article Ahmed, I was lookin...</div>	<div>A GUIDE TO SETTING UP LET'S ENCRYPT SSL ON...</div> <div> Chirag Bhansali 6 Sep</div> <div>What is a zone editor? I couldn't find one in...</div>	<div>HOW TO MASTER YOUR API WORKFLOW WITH...</div> <div> Steve Griffith 18 Aug</div> <div>I actually posted an introductory video...</div>	<div>PUTTING THE APP IN PROGRESSIVE WEB APP...</div> <div> Dev Agrawal 9 Sep</div> <div>Excellent guide... definitely needed thi...</div>	<div>10 LANGUAGES T COMPIL TO JAV...</div> <div> Michel P 25 Aug</div> <div>F# fable is mis:</div>
--	---	--	---	--	--	---



Add a comment...



**Andre Heijstek**  
Thanks a lot!  
I noticed that the edit.html.erb is missing from the instruction.  
Reply · Share ·



**Ilya Bodrov** → Andre Heijstek  
Oh yeah, thank you for noticing that. Will be added soon!  
Reply · Share ·



**Sherif Elkassaby** → Ilya Bodrov  
i'm waiting for the edit view :)  
Reply · Share ·



**Ilya Bodrov** → Sherif Elkassaby  
<https://github.com/bodrovis/Sitepoint-source/blob/master/NestedForms/app/views/places/edit.html.erb> Sorry guys I do not have a write access to this article  
Reply · Share ·

Show 1 more replies



**Jiongye Li**  
Thanks Ilya!  
I also built a gem with a similar idea a few years ago: [https://github.com/jiongye/dynamic\\_fields](https://github.com/jiongye/dynamic_fields)  
Reply · Share ·



**Sherif Elkassaby**  
Thanks for this tutorial it really helps..  
Reply · Share ·



**olegnikitashin**  
Great tutorial as always! Thanks)  
Reply · Share ·



**Ilya Bodrov** → olegnikitashin  
My thanks!  
Reply · Share ·



**Pierre Roy**  
//= require jquery-ui/effect-highlight does not work anymore since last updates  
New syntax is //= require jquery-ui/effects/effect-highlight  
Reply · Share ·



**Ilya Bodrov** → Pierre Roy  
Yeah, true, it has changed in v6. :)  
Reply · Share ·



**Jose V**  
Very well written tutorial! However I seem to be getting an ActiveRecord::RecordNotFound error, whenever I go back and try to edit or add "addresses". Couldn't find Addresses with ID=1 for Place with ID=  
  
Any suggestions?  
Reply · Share ·



**Ilya Bodrov** → Jose V  
Thank you! Well, it seems you have some issues with params. Check that they are really sent properly and have proper names. It can be checked from the console or by using pry-rails gem to stop code execution  
Reply · Share ·



**Jose V** → Ilya Bodrov  
Thanks for the response! Will definitely take another look at my params.  
Reply · Share ·

## Stuff We Do

- [Premium \(/premium/\)](/premium/)
- [Versioning \(/versioning/\)](/versioning/)
- [Themes \(/themes/\)](/themes/)
- [Forums \(/community/\)](/community/)
- [References \(/html-css/css/\)](/html-css/css/)

## About

- [Our Story \(/about-us/\)](/about-us/)
- [Press Room \(/press/\)](/press/)

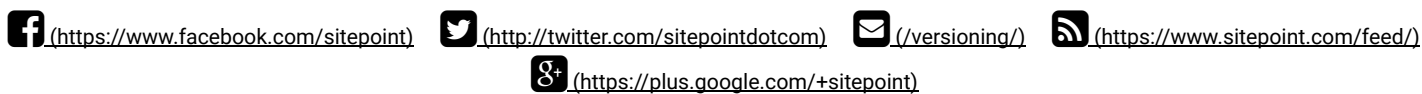
## Contact

- [Contact Us \(/contact-us/\)](/contact-us/)
- [FAQ \(https://sitepoint.zendesk.com/hc/en-us\)](https://sitepoint.zendesk.com/hc/en-us)
- [Write for Us \(/write-for-us/\)](/write-for-us/)
- [Advertise \(/advertise/\)](/advertise/)

## Legals

- [Terms of Use \(/legals/\)](/legals/)
- [Privacy Policy \(/legals/#privacy\)](/legals/#privacy)

## Connect



© 2000 – 2017 SitePoint Pty. Ltd.

Recommended Hosting Partner:  [SiteGround \(https://www.siteground.com/go/sitepoint-siteground-promo\)](https://www.siteground.com/go/sitepoint-siteground-promo)

