Episode 255

Undo with PaperTrail

Confirmation dialogue boxes are common in web applications. Almost every Rails application shows one when we click a “delete” link, asking if we’re sure we want to delete an item. Most of the time these alerts are unnecessary, we clicked the link because we want the item to be deleted, but there are, of course, times when we click a link by mistake. Wouldn’t it be better, though, if there was no confirmation but instead a link offering the ability to undo the change we just made? This would provide a much smoother user experience.

In this episode we’ll be implementing this behaviour by making use of a gem called PaperTrail[1] which is a generic versioning library for ActiveRecord. There are specific undo frameworks for Rails, for example Vestal Versions[2] which was covered back in episode 177 [watch[3], read[4]]. We’ll use PaperTrail here as it works better when adding undo support.

Installing PaperTrail

To install PaperTrail we need to add a reference to it to our application’s `Gemfile` and then run the `bundle` command to install it.

`/Gemfile`

```
# Edit this Gemfile to bundle your application's dependencies.
source 'http://gemcutter.org'

gem "rails", "3.0.5"
gem "sqlite3-ruby", :require => "sqlite3"
gem "paper_trail"
```

When the gem has installed we’ll can run the PaperTrail install generator.

```
$ rails g paper_trail:install
```

This generator creates a migration file that will create a `versions` table once we’ve run `rake db:migrate`.

---

[1] https://github.com/airblade/paper_trail

[2] https://github.com/laserlemon/vestal_versions

[3] http://railscasts.com/episodes/177-model-versioning

[4] http://asciicasts.com/episodes/177-model-versioning

Using PaperTrail

To add versioning to a model, such as this application's `Product` model we add a call to `has_paper_trail` in the model file.

/app/models/product.rb

```
class Product < ActiveRecord::Base
  attr_accessible :name, :price, :released_at
  has_paper_trail
end
```

The `Product` model will now be versioned so that any changes made can be undone.

Adding Undo Functionality

Getting PaperTrail installed is simple enough but how do we go about adding undo functionality to our application? To start with we'll need a controller action for the undo link to point to. We could add a new action to the `ProductsController`, but to keep the code cleaner we'll create a new controller called `versions`. This approach means that we can more easily add versioning to other models later.

```
$ rails g controller versions
```

We only need one action in this controller, to revert a version.

/app/controllers/versions_controller.rb

```
class VersionsController < ApplicationController
  def revert
    @version = Version.find(params[:id])
    @version.reify.save!
    redirect_to :back, :notice => "Undid #{@version.event}"
  end
end
```

In this action we fetch the `Version` that matches the `id` parameter that is passed in from the URL. We then want to get the model object that is specific to that version

and we do that by calling `reify` on the version. In this case this will return the `Product` instance that the version refers to. We can then call `save!` on this to revert that product back to that version. Once we've saved that version we'll redirect back to the previous page and set a flash message to say what's just happened. We can get the event that has just completed by calling `@version.event`. This will return either "create", "update" or "destroy" depending on the action that has just been reverted and we'll put that in the message.

We'll need to be able to access this new action so we'll add a new route to the `routes` file.

/config/routes.rb

```
Store::Application.routes.draw do |map|
  post "versions/:id/revert" => "versions#revert", :as => ↵
   "revert_version"
  resources :products
  root :to => "products#index"
end
```

Note that we've used the `post` method here. We're doing something potentially destructive in the `revert` action as it alters the database so we won't use `match` as this will accept GET requests. As the name implies `post` only responds to POST requests.

Undoing Updates

Now that we have an action that will handle undo behaviour it's time to add a link to the flash notices inside the `ProductsController`. We'll start with the `update` action so that when someone updates a product they can click a link and undo the changes they've made.

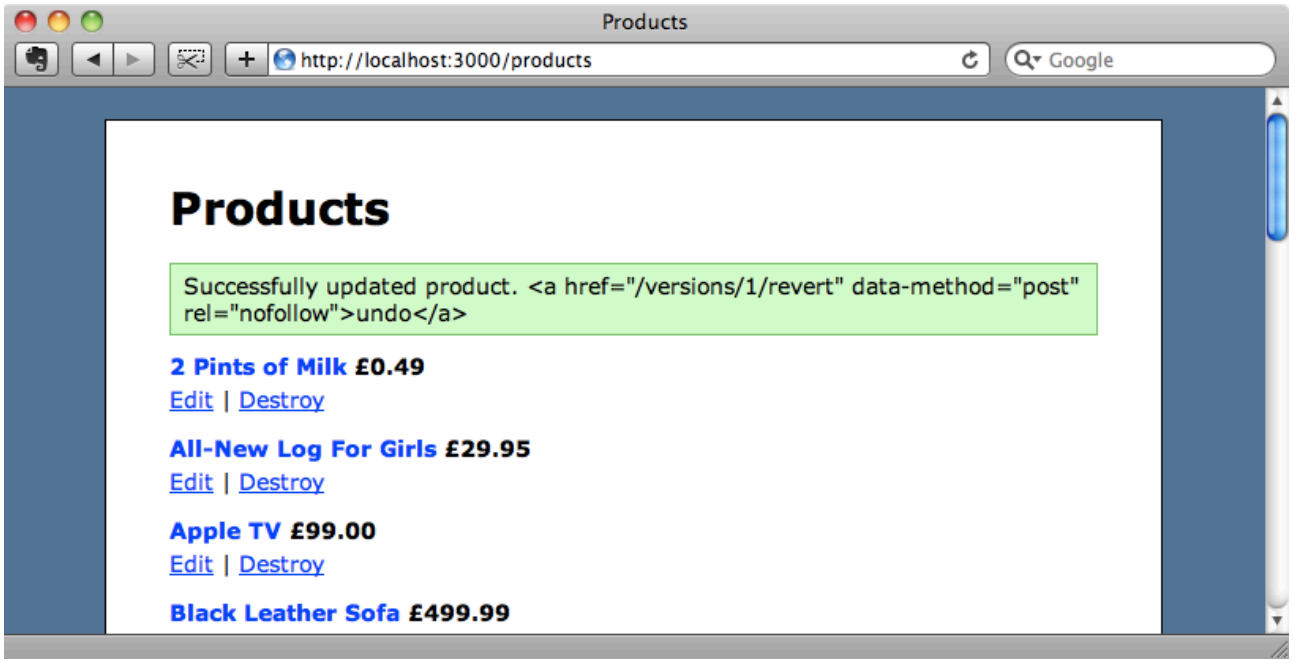
We'll need to create the undo link, but how should we do that in the controller? We could put some HTML into a string but it would be much nicer if we could use `link_to` in the controller. We can't use view methods like `link_to` directly in controllers, but we can access them through the `view_context` so we can call `view_context.link_to` to create a link. This link will point to the `revert` action and pass in the `id` of last stored version of that product to it. We can call

`@product.versions` to get all of the versions of a product and call `last` on that to get the most recent stored version. Knowing this we can now create the undo link. Note that it has `:method => :post` set on it as the `revert` action will only respond to POST requests. Having constructed the link we can add it to the flash message.

/app/controllers/products_controller.rb

```
def update
  @product = Product.find(params[:id])
  if @product.update_attributes(params[:product])
    undo_link = view_context.link_to("undo", ↵
      revert_version_path(@product.versions.last), ↵
      :method => :post)
    redirect_to products_url, :notice => ↵
      "Successfully updated product."
  else
    render :action => 'edit'
  end
end
```

We have enough code in place now to try this out. If we edit one of the products in the list, for example changing “1 Pint of Milk” to “2 Pints of Milk”, we’ll get the following result.



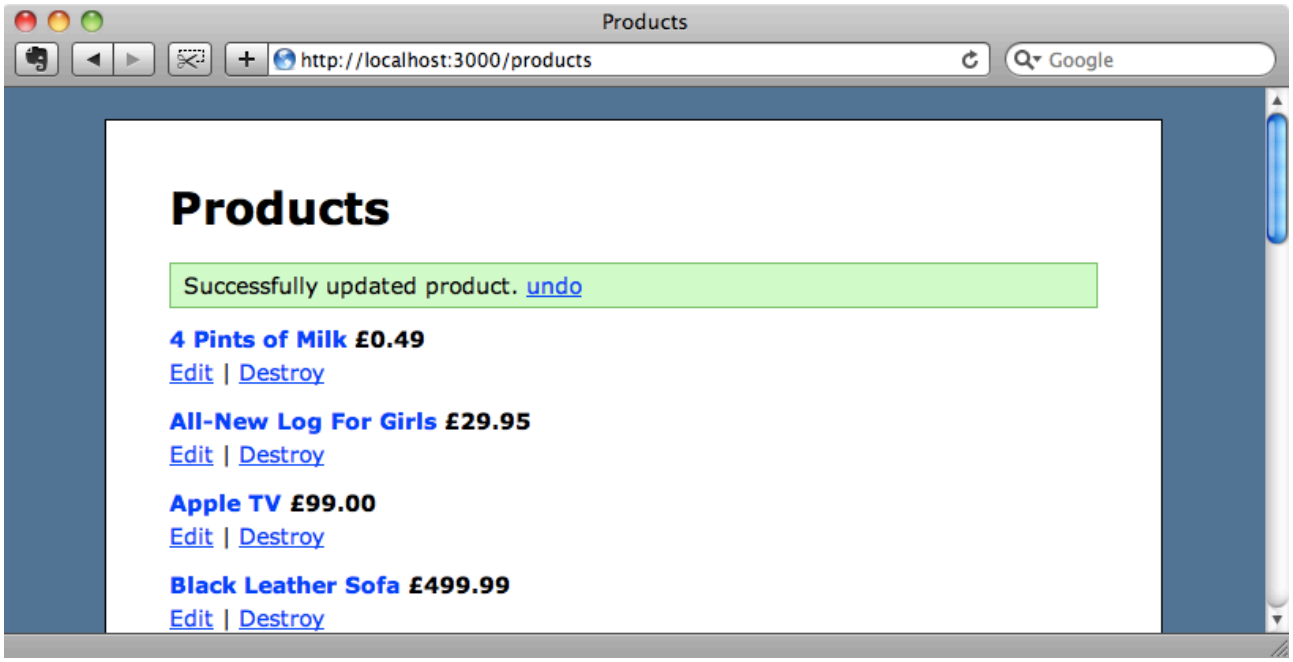
The product has changed but the link hasn't quite worked. The link is there, but its HTML has been escaped. To fix this we'll need to go to the place in the application that renders the flash messages and modify it so that it doesn't escape their contents. In this application this takes place in the layout file. All we need to do here is wrap the message in the `raw` method so that its content isn't escaped.

/app/views/layouts/application.html.erb

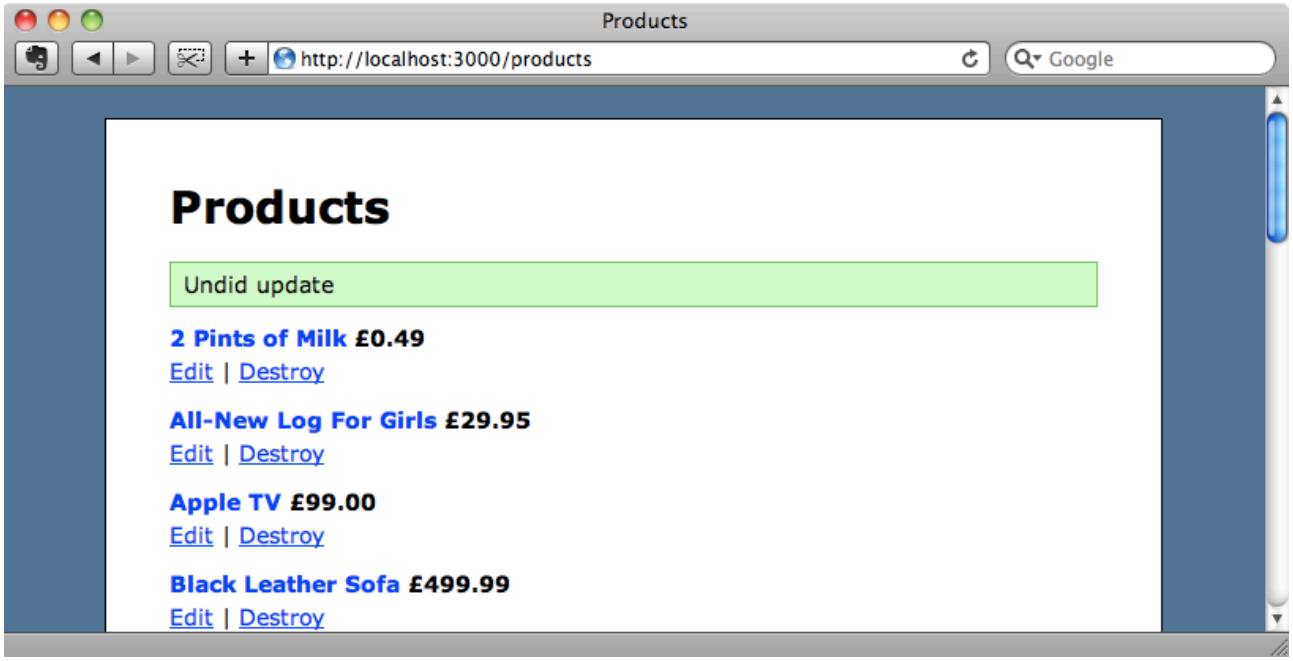
```
<div id="container">
  <h1><%=h yield(:title) %></h1>
  <%- flash.each do |name, msg| -%>
    <%= content_tag :div, raw(msg), :id => "flash_#{name}" %>
  <%- end -%>
  <%= yield %>
</div>
```

We have to be careful when doing this. If any input from the user is displayed in the flash message it won't be escaped and we'll have to remember to escape it before displaying it.

If we edit the product again, changing it to "4 Pints of Milk" this time the undo link is now shown correctly.



When we click the undo link the product is reverted back to its previous version.



Undoing After Deleting an Item

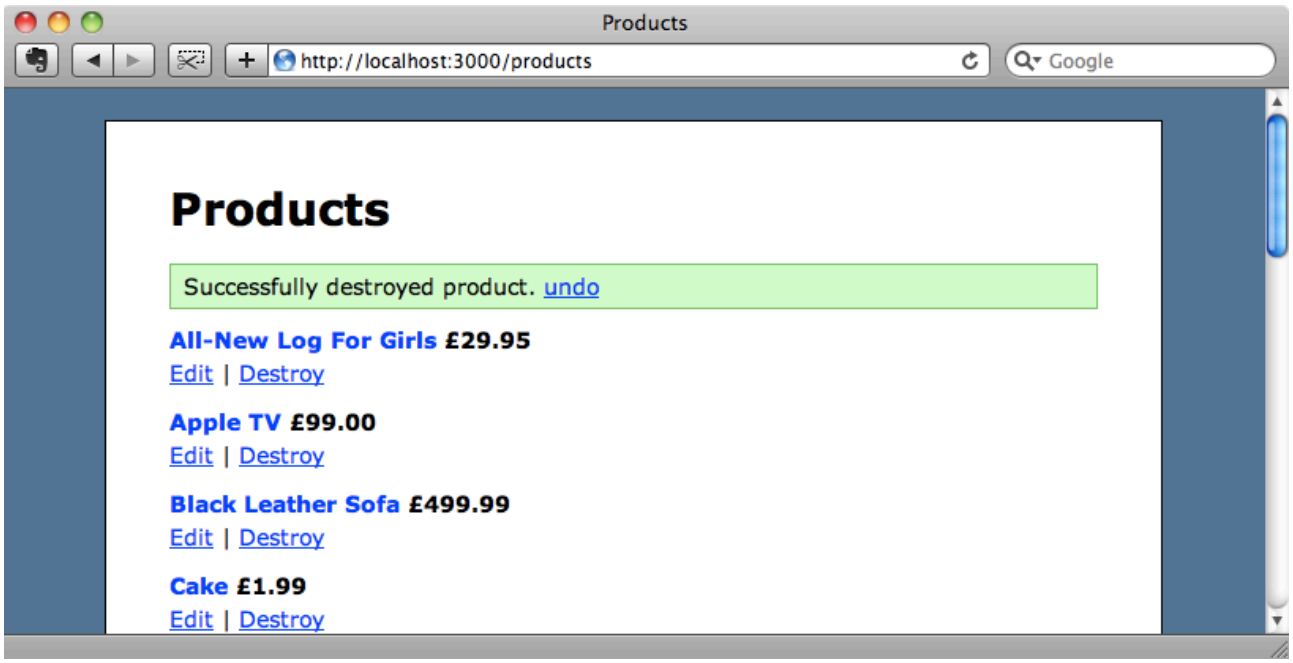
Next we'll add an undo link to the `destroy` action so that we can reinstate an item after deleting it. We'll be building up the link in the same way we did in the `update` action so the first thing we'll do is move the code that creates the link into a separate method that we can then call from both `update` and `destroy`. We'll make it a private method so that it isn't considered an action and call it `undo_link`.

/app/controllers/products_controller.rb

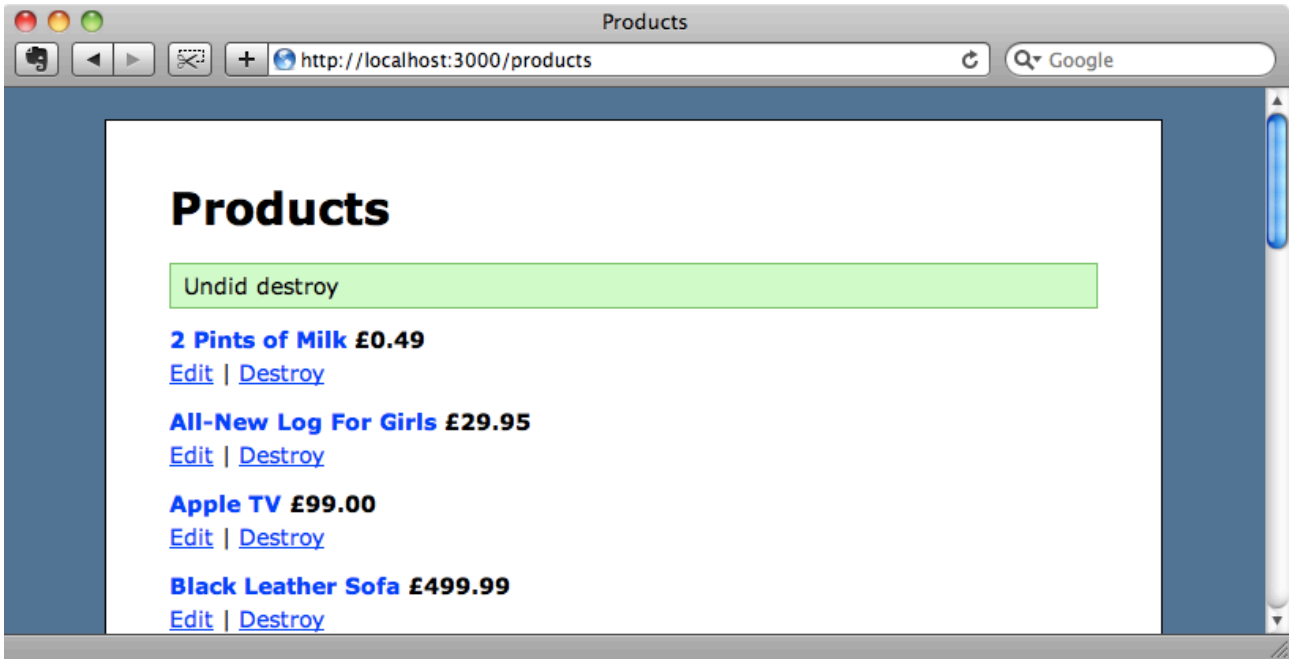
```
class ProductsController < ApplicationController

  #other actions omitted.

  def update
    @product = Product.find(params[:id])
    if @product.update_attributes(params[:product])
      redirect_to products_url, ↵
        :notice => "Successfully updated product. #{undo_link}"
    else
      render :action => 'edit'
    end
  end

  def destroy
    @product = Product.find(params[:id])
    @product.destroy
    redirect_to products_url, ↵
      :notice => "Successfully destroyed product. #{undo_link}"
  end

  private
  def undo_link
    view_context.link_to("undo", ↵
     revert_version_path(@product.versions.scoped.last), ↵
     :method => :post)
  end
end
```

There is one small gotcha here. When we're deleting a record `product.versions` still refers to a list of versions that doesn't include the most recent version (the version that was deleted by the `destroy` model). It seems that the versions are being cached in an array. In this case we should be able to call `@product.versions(true)` to flush the cache as we can with normal associations but this doesn't work like we'd expect when deleting records. To get around this we can call `scoped` on the versions array before calling `last` and this will mean that we're always referring to the most recent version.

Let's test this out. If we click the "Destroy" link for "2 Pints of Milk" and confirm the deletion then that item is removed.



When we click the "undo" link the deleted item is reinstated.



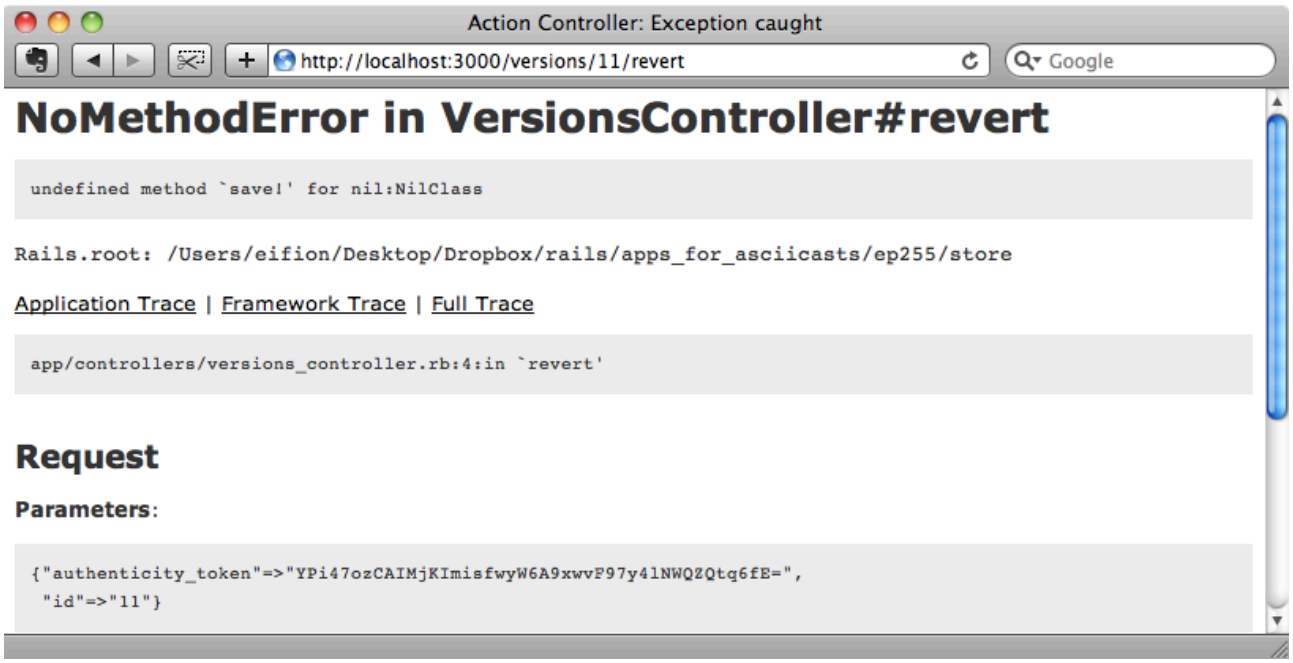
Undoing a Create

There's one action left to implement: `create`. Let's try adding the undo link to the flash message in the `create` action as we did with `update` and `destroy` and see how we get on.

/app/controllers/products_controller.rb

```
def create
  @product = Product.new(params[:product])
  if @product.save
    redirect_to products_url, :notice => "Successfully created ↵
      product. #{undo_link}"
  else
    render :action => 'new'
  end
end
```

When we add a new product now and try to undo it, however, we get an error message.



The code here is trying to save a record but we're trying to undo the creation of a product here so what we really want to do is destroy the created item rather than trying to save it. We need to modify the code in the `VersionsController` and change its behaviour.

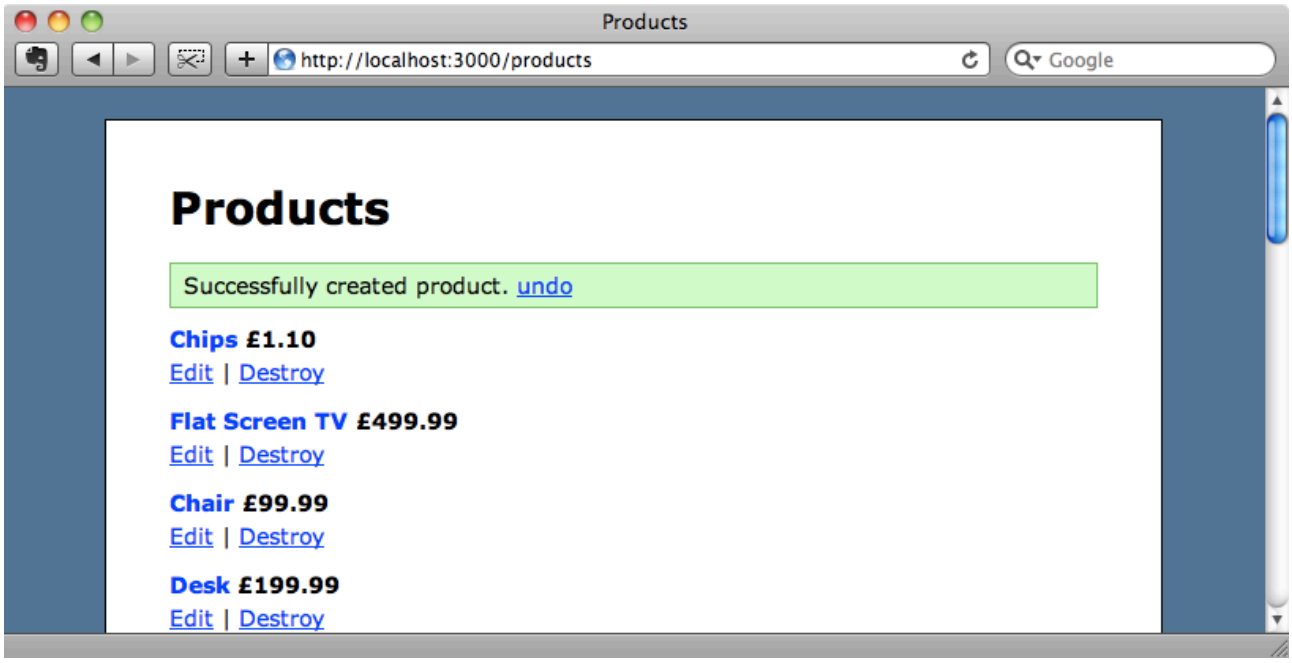
The problem is that when we call `reify` on a newly-created item it returns `nil` as there is no previous version. We need to modify the code in the `revert` action so that it checks to see if that previous version exists and, if so, saves it. Otherwise it will delete the item.

/app/controllers/versions_controller.rb

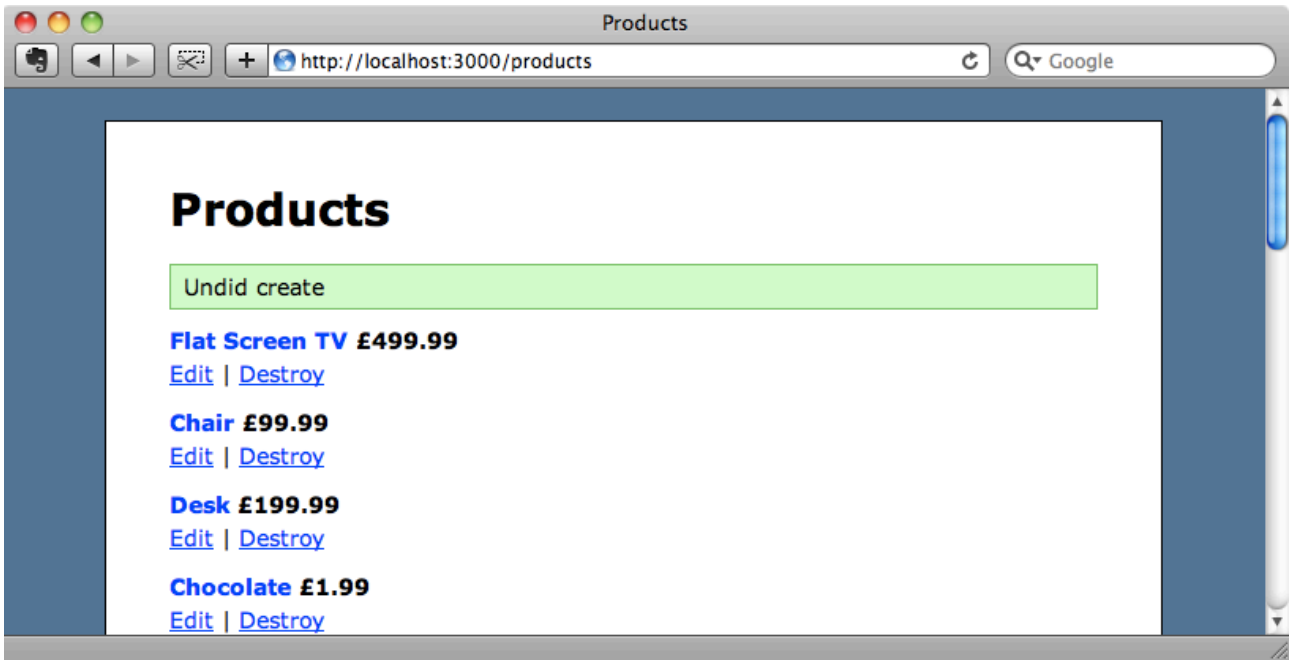
```
class VersionsController < ApplicationController
  def revert
    @version = Version.find(params[:id])
    if @version.reify
      @version.reify.save!
    else
      @version.item.destroy
    end
    redirect_to :back, :notice => "Undid #{@version.event}"
  end
end
```

If no previous version exists then we call `@version.item` which will return the newly created `Product` from the products table. We can then call `destroy` on that to delete it.

Let's give it a try. We'll create a new product called "Chips".



When we click the “undo” link the newly-created item is removed.



Redoing The Undo

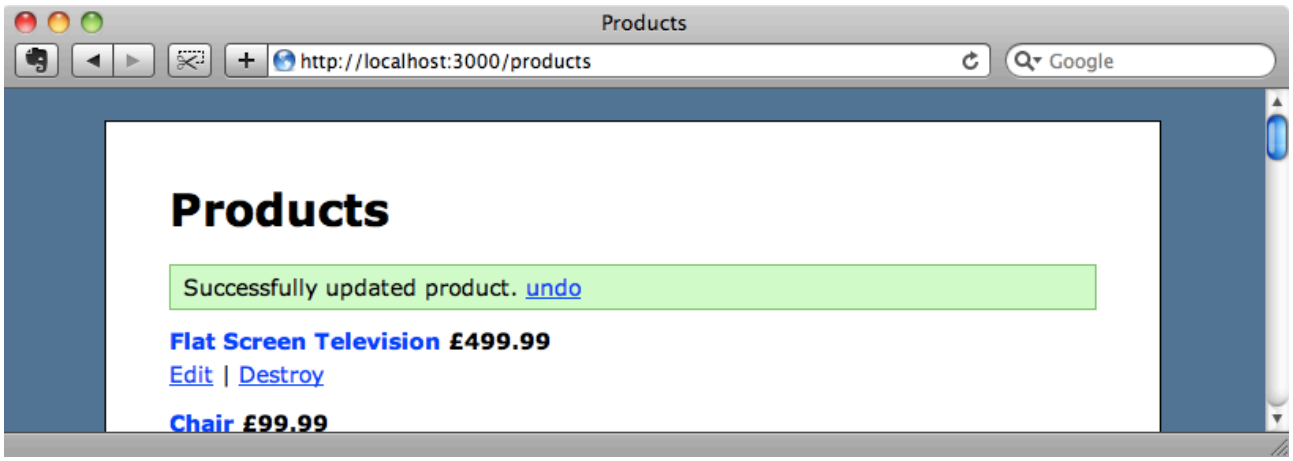
Now that we can undo after creating, updating or destroying a product it would be useful to add the ability to redo the action we’ve just undone. To add redo behaviour we have to go into our `VersionsController` and add a link to the flash message that shows when we undo an action.

/app/controllers/versions_controller.rb

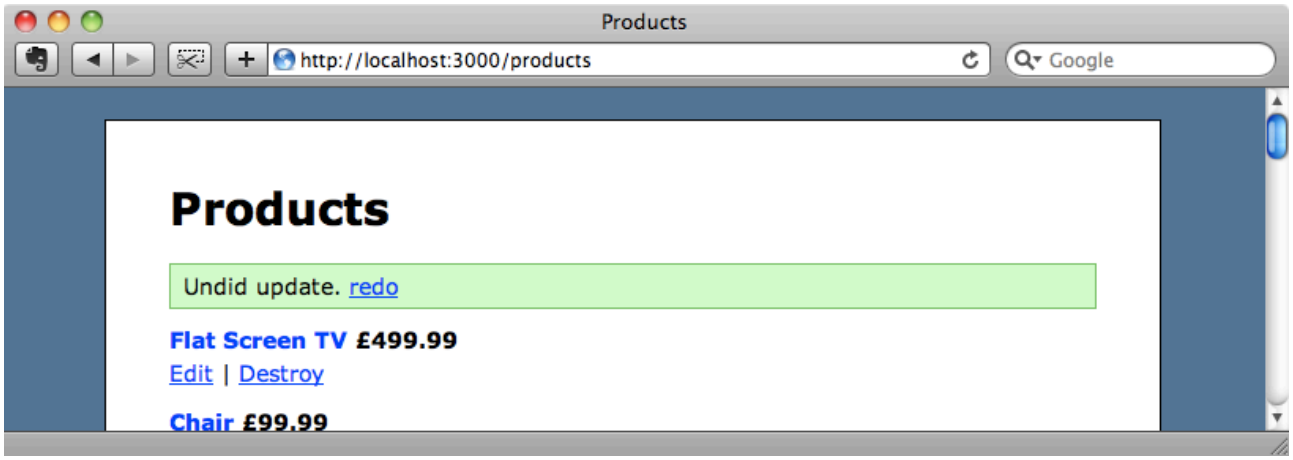
```
class VersionsController < ApplicationController
  def revert
    @version = Version.find(params[:id])
    if @version.reify
      @version.reify.save!
    else
      @version.item.destroy
    end
    link_name = params[:redo] == "true" ? "undo" : "redo"
    link = view_context.link_to(link_name, ↵
      revert_version_path(@version.next, ↵
      :redo => !params[:redo]), :method => :post)
    redirect_to :back, :notice => ↵
     "Undid #{@version.event}. #{link}"
  end
end
```

Most of the logic here is to do with changing the text in the link depending on what the previous action was. Then we generate the link which uses `revert_version_path` and which points to the next version of the item in the version chain. The reason it chooses the next version is because when we save or destroy an item it will create a new version record and that is the one we want to go to get the redo behaviour.

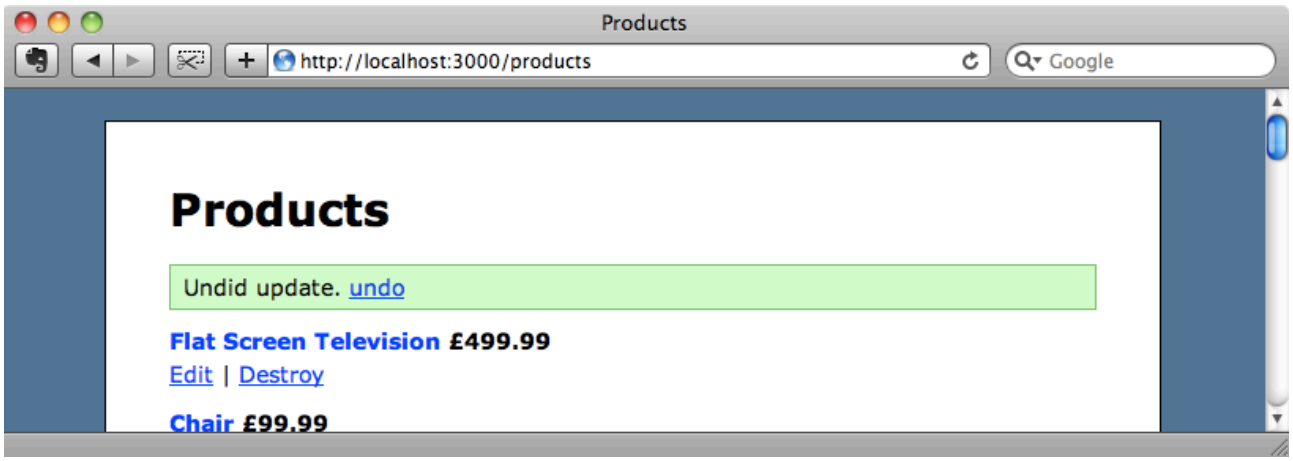
We can try this out now. We'll edit "Flat Screen TV" and change its name to "Flat Screen Television". As expected this gives us an undo link.



When we click the undo link the product reverts back as before but now we now have a redo link too.



Clicking the redo link redoes the change and the title changes back to the modified version.



We can toggle between “undo” and “redo” as often as we like and the item will swap between its last two versions. Now that we have full undo and redo capabilities we can finally remove the confirmation dialog from the destroy links.

Managing Old Versions

As we use this application now the data in the `versions` table will build up with a lot of information. Storing all of the version data in one database table makes it easy to remove old versions with a command like this:

```
Version.delete_all["created_at < ?", 1.week.ago]
```

We can place the above command in a `rake` task and call it in a regular basis using the Whenever gem. Take a look at episode 164 [watch[5], read[6]] for more information about how to do that.

Another great feature of PaperTrail is that it makes it easy to store additional information in the versions table. All we need to do is add a new column to the `versions` table. We can then use the `:meta` option of the `has_paper_trail` method or use the `info_for_paper_trail` method in the controller and supply the additional options there. If we want to add some additional information in the versions table, such as the name of the model that's modified so that it can be shown in the flash message we can add that information here and display it in the `revert` action.

---

[5] http://railscasts.com/episodes/164-cron-in-ruby

[6] http://asciicasts.com/episodes/164-cron-in-ruby