Extracted from:

Rails Recipes

Tagging Your Content

Problem

By now, it's a fairly safe bet that you (and many of the users of your software) have heard of this thing called *social networking*. It was recently all the rage. Cutting edge. A delight to use and a differentiator for applications that used it.

Now, though, it's *expected* that web applications will employ some kind of social networking effect where relevant. If you're looking for books, you expect the online bookstore to leverage the shopping behavior of the masses to help you find books you might like. Or music. Or whatever you might be trying to do or explore. And, though it's possible to hire computer scientists to develop algorithms for predicting what each user is going to be looking for, it's a lot cheaper and easier to let your users do the work.

So after the dust has settled, the heart of what's left in the "social" applications arena is *tagging*. You put simple, textual, nonhierarchical identifiers on items in an application, and the cumulative effect of all the application's users doing this creates a self-organizing system. It's an idea made popular by sites like del.icio.us and Flickr that has now taken over the Web. If you're lucky, tags on your site will help users find new favorite things they didn't even know they liked.

So, how do we do this in Rails?

Ingredients

- David Heinemeier Hansson's acts_as_taggable plugin, installable from the root of your Rails application with the following:

```
chad> ruby script/plugin install acts_as_taggable
+ ./acts_as_taggable/init.rb
+ ./acts_as_taggable/lib/README
+ ./acts_as_taggable/lib/acts_as_taggable.rb
+ ./acts_as_taggable/lib/tag.rb
+ ./acts_as_taggable/lib/tagging.rb
+ ./acts_as_taggable/test/acts_as_taggable_test.rb
```

- Rails 1.1 or higher. acts_as_taggable() relies on polymorphic associations, a feature added after Rails 1.0 (see Recipe 23, *Polymorphic Associations—has_many :whatevers*, on page 96).

Solution

Assuming you have already installed the acts_as_taggable plugin, the first step in adding tagging to your application is to set up the database to hold the tags and their associations with your models. The migration for the database should look something like the following:

Download Tagging/db/migrate/001_add_database_structure_for_tagging.rb

```
class AddDatabaseStructureForTagging < ActiveRecord::Migration
  def self.up
    create_table :taggings do |t|
      t.column :taggable_id, :integer
      t.column :tag_id, :integer
      t.column :taggable_type, :string
     end
     create_table :tags do |t|
      t.column :name, :string
     end
  end

  def self.down
    drop_table :taggings
    drop_table :tags
  end
end
```

So now we have the ability to tag something. We just need something to tag! Let's make a simple model to tag. In your own application, of course, you'd already have models to tag. But for the sake of demonstration, let's create a simple model for tracking contacts in an address book application. As always, we'll use an Active Record migration:

Download Tagging/db/migrate/002_add_contacts_table.rb

```
class AddContactsTable < ActiveRecord::Migration
  def self.up
    create_table :contacts do |t|
      t.column :name, :string
      t.column :address_line1, :string
      t.column :address_line2, :string
      t.column :city, :string
      t.column :state, :string
      t.column :postal_code, :string
    end
  end
```

```
  def self.down
    drop_table :contacts
  end
end
```

Next we'll generate the model and make it taggable. We don't need to create models for the actual Tag objects, because they're included in the acts_as_taggable plugin.

Download Tagging/app/models/contact.rb

```
class Contact < ActiveRecord::Base
  acts_as_taggable
end
```

Believe it or not, we now have taggable contacts. Let's look in the console:

```
chad> ruby script/console
Loading development environment.
>> c = Contact.create(:name => "Josef K", :address_line1 => "123 Main St.",
    :address_line2 => "Apt. 2", :city => "Vienna",
    :state => "Colorado", :postal_code => "54321")
=> #<Contact:0x267a8f8 @new_record=false, @base=#<Contact:0x267a8f8 ...>>
>> c.tag_with("friends colorado existentialists")
=> ["friends", "colorado", "existentialists"]
```

Here we created an instance of Contact and used the tag_with() method to tag it with a space-delimited list of tags. The acts_as_taggable plugin automatically parses the list and either creates new Tag instances or associates existing ones. The associated tags are then available via the tags() method on the model:

```
>> c.tags
    => [#<Tag:0x264f450 @attributes={"name"=>"friends", "id"=>"1"}>,
        #<Tag:0x264f414 @attributes={"name"=>"colorado", "id"=>"2"}>,
        #<Tag:0x264f3d8 @attributes={"name"=>"existentialists", "id"=>"3"}>]
```

Now if we were to create a new contact and tag it with an already existing tag, we'll see that the existing instance of the tag in the database is reused and associated with the model:

```
>> c2 = Contact.create(:name => "John Barth", :address_line1 => "432 South End Rd.",
      :city => "Gotham", :state => "North Carolina", :postal_code => "12345")
=> #<Contact:0x26463c8 @new_record=false, @base=#<Contact:0x26463c8 ...>>
>> c2.tag_with("friends carolina pragmatists")
=> ["friends", "carolina", "pragmatists"]
>> c2.tags
=> [#<Tag:0x2605bc0 @attributes={"name"=>"friends", "id"=>"1"}>,
    #<Tag:0x2605b84 @attributes={"name"=>"carolina","id"=>"4"}>,
    #<Tag:0x2605b48 @attributes={"name"=>"pragmatists", "id"=>"5"}>]
```

OK. Our models are ready to be tagged! Let's get our heads out of the console and put the tags to use on a real web application. Most tag-enabled applications will want to do three tasks: assign tags to an item, view an item's tags, and search for items by tag. We'll start with the easiest part: viewing an item's tags.

The first thing we need is the ability to actually view an item, so we'll whip up a simple action for that. The following is the beginning of our ContactsController class:

Download Tagging/app/controllers/contacts_controller.rb

```
class ContactsController < ApplicationController
  def list
    @contacts = Contact.find(:all)
  end
```

This is a typical list action. We'll get a little fancier with the view and throw in some user-friendly Ajax effects. After all, these days tagging without Ajax is like wearing a mink coat with an old, worn-out pair of tennis shoes. Our contacts/list.rhtml is a simple wrapper for a partial template that contains the real display logic for our contacts:

Download Tagging/app/views/contacts/list.rhtml

```
<ul id="contacts-list">
  <% if @contacts.blank? %>
    <li class="no-contacts">No contacts to display</li>
  <% else %>
  <%= render :partial => "detail", :collection => @contacts %>
  <% end %>
</ul>
```

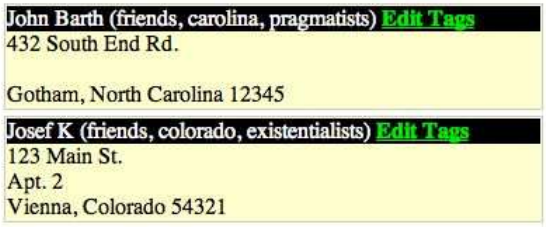
We use a partial template because it separates the code into smaller more manageable chunks and also because we're going to use the same partial view as the rendered response of our Ajax requests. The template contacts/_detail.rhtml consists of two parts: the contact display and a form for editing a contact's tags. To support subsequent Ajax requests, the display part is separated into another partial template, contacts/_content.rhtml:

Download Tagging/app/views/contacts/_content.rhtml

```
<div class="name"><%= contact.name %>
  <%=
     if contact.tags.blank?
       ""
     else
       "(" + contact.tags.collect{|tag| tag.name}.join(", ") + ")"
     end
```

```
  %>
  <%= link_to_function("Edit Tags", "Element.toggle($('#{form_id}'))") %>
</div>
<div class="address">
  <%= contact.address_line1 %><br/>
  <%= contact.address_line2 %><br/>
  <%= contact.city %>, <%= contact.state %>  <%= contact.postal_code %>
</div>
```

This is mostly typical display code. We display a contact's tags, if any, in parentheses next to the contact's name. Here's what it looks like in the browser:



Each contact gets its own separate form for editing the contact's tags. This form starts out hidden and is displayed via the Element.toggle() JavaScript call when a user clicks the "Edit Tags" link. Completing the contact display implementation, here's the full contacts/_detail.rhtml code that creates the form for editing a contact's tags and references the display partial:

Don't forget to include the necessary JavaScript files for the Ajax effects to work. Somewhere in the <*head*> section of your view, you'll need this:

```
<%= javascript_include_tag :defaults %>
```

Download Tagging/app/views/contacts/_detail.rhtml

```
<li class="contact-item">
<% form_id = "tag-form-for-#{detail.id}" %>
<%= form_remote_tag :url => {:action => "tag", :id => detail},
                    :complete => "Element.toggle($('#{form_id}'))",
                    :success => visual_effect(:shake, "contact-#{detail.id}"),
                    :update => "contact-#{detail.id}",
                    :html => {:id => form_id, :style => "display:none"} %>

  <%= text_field_tag "tag_list",
                     detail.tags.collect{|t| t.name}.join(" "),
                     :size => 40 %>

  <%= hidden_field_tag "form_id", form_id %>
  <%= submit_tag "save" %>
<%= end_form_tag %>
```

```
<div id="contact-<%=detail.id%>" class="contact-details">
<%= render :partial => "content",
           :locals => {:contact => detail, :form_id => form_id} %>
</div>
</li>
```

We first generate an HTML ID for the form, which we use to toggle the form's display on and off. Then, since we want tag updates to be as painlessly easy as possible, we create the form via form_remote_tag(). When a user submits the form, it will make an asynchronous HTTP request in the background to the tag action of our ContactsController. On successful completion of that request, the tag form will be toggled closed, the contact display will be updated, and we'll give the contact's display a little shake to let the user know something happened.

All that's left to actually make tagging happen is to implement the tag action. We already learned how to do this in our script/console session earlier, so the implementation is easy:

Download Tagging/app/controllers/contacts_controller.rb

```
def tag
  contact = Contact.find(params[:id])
  contact.tag_with(params[:tag_list])
  contact.save
  render :partial => "content",
         :locals => {:contact => contact, :form_id => params[:form_id]}
end
```

Now that we can display and edit a contact's tags, all we lack is the ability to search for a contact by tag. Since we already created the list() action, it makes sense to modify it for our needs instead of creating yet another action that displays a list. Here's the revised version of our action:

Download Tagging/app/controllers/contacts_controller.rb

```
def list
  @contacts = if tag_name = params[:id]
    Tag.find_by_name(tag_name).tagged
  else
    Contact.find(:all)
  end
end
```

This code reads a tag name supplied in the URI and finds items tagged with that name. So, for example, you could call the application with the URI /contacts/list/colorado to list only those contacts tagged with colorado.

If no tag is supplied on the URI, it returns a list of *all* the contacts in the database as before.

A nice feature of the acts_as_taggable() library is that you can use it to tag more than one model type. For example, let's say our little contact database were to blossom into a full-blown personal information manager and we added the ability to create both notes and calendar appointments. Naturally, it would make sense to tag these features along with our contacts.

Because acts_as_taggable() uses Active Record's new polymorphic associations feature, we can tag any model we'd like. All we need to do is declare each model as acts_as_taggable(), and the plugin takes care of the rest for us.

Discussion

In our schema, we haven't yet added any database indexes. For a large application, it would make sense to create indexes on various fields in the supplied tables, including but not limited to the name column of the tags table.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help programmers stay on top of their game.

Visit Us Online

Rails Recipes Home Page
`pragmaticprogrammer.com/titles/fr_rr`
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
`pragmaticprogrammer.com/updates`
Be notified when updates and new books become available.

Join the Community
`pragmaticprogrammer.com/community`
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
`pragmaticprogrammer.com/news`
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragmaticprogrammer.com/titles/fr_rr.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragmaticprogrammer.com/catalog
Customer Service:	support@pragmaticprogrammer.com
Non-English Versions:	translations@pragmaticprogrammer.com
Pragmatic Teaching:	academic@pragmaticprogrammer.com
Author Proposals:	proposals@pragmaticprogrammer.com