Extracted from:

Rails Recipes

Creating a Drag-and-Drop Sortable List

Credit

Thanks to Bruce Williams for code that inspired this recipe.

Problem

Your application has a model with a list that should be sortable. You want to manage the sort order of the list in the database, and you want to give your users a snazzy, modern, drag-and-drop interface with which to maintain the sort order.

Solution

Let's say we're creating an application for managing grocery lists. With the size of today's American grocery superstore, it's important to devise a shopping strategy before you hit the aisles. Otherwise, you can waste precious hours of your life following unoptimized shopping routes.

The Active Record migration file for our shopping optimization application will look like the following:

Download DragAndDropSortableList/db/migrate/001_add_person_and_grocery_lists_and_food_items_tables.rb

```ruby
class AddPersonAndGroceryListsAndFoodItemsTables < ActiveRecord::Migration
  def self.up
    create_table :people do |t|
      t.column :name, :string
    end

    create_table :grocery_lists do |t|
      t.column :name, :string
      t.column :person_id, :integer
    end

    create_table :food_items do |t|
      t.column :grocery_list_id, :integer
      t.column :position, :integer
      t.column :name, :string
      t.column :quantity, :integer
    end
  end
  def self.down
    drop_table :people
    drop_table :grocery_lists
    drop_table :food_items
  end
end
```

As you can see, we have tables to support people, their grocery lists, and the items that go on each list (along with the quantity we need of each item). This is all standard Active Record has_many() fare, except for the position column in the food_items table. This column is special, as we'll see in a moment.

The associated model files are similarly short and sweet. A Person has many GroceryList objects:

Download DragAndDropSortableList/app/models/person.rb

```
class Person < ActiveRecord::Base
  has_many :grocery_lists
end
```

And each GroceryList has a list of FoodItem objects on it, which will be retrieved by the food_items table's position column:

Download DragAndDropSortableList/app/models/grocery_list.rb

```
class GroceryList < ActiveRecord::Base
  has_many :food_items, :order => :position
  belongs_to :person
end
```

Finally, we get to the spice. Class FoodItem contains Active Record's acts_as_list() declaration, which allows its containing object (GroceryList) to "automagically" manage its sort order:

Download DragAndDropSortableList/app/models/food_item.rb

```
class FoodItem < ActiveRecord::Base
  belongs_to :grocery_list
  acts_as_list :scope => :grocery_list
end
```

The :scope parameter tells acts_as_list() that the sort order is relevant within the context of a single grocery_list_id. This is so one grocery list's sort order doesn't affect any other list's order.

The column name position is special to acts_as_list(). By convention, Rails will automatically use this column name to manage sort order when a model is declared acts_as_list(). If we needed to use a nonstandard column name here, we could have passed the :column parameter, but position makes sense for our humble grocery list manager, so we'll leave well enough alone.

After running the migration and creating the model files, let's fire up the Rails console and play with this new structure:

```
chad> ruby script/console
 >> kelly = Person.create(:name => "Kelly")
 => #<Person:0x26ec854 ...>>
 >> list = kelly.grocery_lists.create(:name => "Dinner for Tibetan New Year Party")
 => #<GroceryList:0x26b9788 ...>>
 >> list.food_items.create(:name => "Bag of flour", :quantity => 1)
 => #<FoodItem:0x26a8898 ...>>
 >> list.food_items.create(:name => "Pound of Ground Beef", :quantity => 2)
 => #<FoodItem:0x269b60c ...>>
 >> list.food_items.create(:name => "Clove of Garlic", :quantity => 5)
 => #<FoodItem:0x26937e0 ...>>
```

So we now have a person named Kelly in our database who seems to be planning a party for the Tibetan New Year celebration. So far, she has three items on her list. She's not done with the list yet, obviously—you can't make momos with just these three ingredients! Let's see what happened to that position column when we created these objects:

```
>> list.food_items.find_by_name("Pound of Ground Beef").position
    => 2
    >> list.food_items.find_by_name("Bag of flour").position
    => 1
```

Cool! Active Record has updated the position column for us! acts_as_list() also sets up a bunch of nice convenience methods for performing tasks such as selecting the next item (in order) in the list or moving an item's position up or down. Let's not get all caught up in the model just now, though. We have enough implemented that we can get to the fun stuff—drag and drop!

As always, if you're going to do fancy Ajax stuff, you need include the necessary JavaScript libraries somewhere in your HTML. I usually create a standard layout and throw the JavaScript in there. Let's create the layout in app/views/layouts/standard.rhtml and then fill it in as follows:

Download DragAndDropSortableList/app/views/layouts/standard.rhtml

```
<html>
  <head>
        <%= javascript_include_tag :defaults %>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

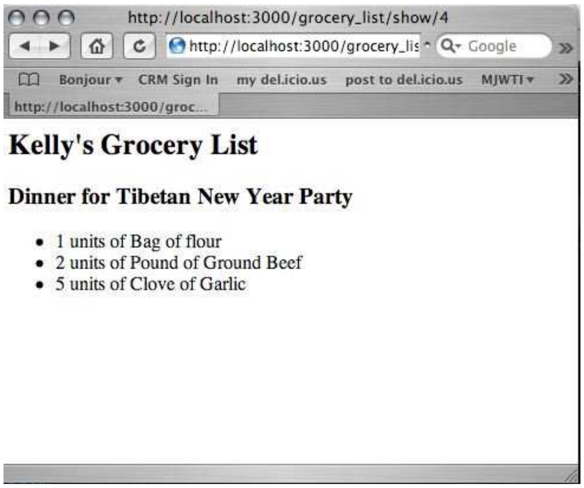
Next, pretending that we already have some kind of interface for creating a list and associating it with a person, let's create the controller and action from whence we'll reorder our list. We'll create a controller in app/controllers/grocery_list_controller.rb with an action called show(). The beginning of the controller should look like the following:

Download DragAndDropSortableList/app/controllers/grocery_list_controller.rb

```
class GroceryListController < ApplicationController
  layout "standard"

  def show
    @grocery_list = GroceryList.find(params[:id])
  end
  # ...
```

Note that we've included the standard.rhtml layout, and we've defined a basic action that will simply find a grocery list based on a supplied parameter:


http://localhost:3000/grocery_list/show/4
http://localhost:3000/grocery_lis
Google
Bonjour
CRM Sign In
my del.icio.us
post to del.icio.us
MJWTI
http://localhost:3000/groc...

Kelly's Grocery List

Dinner for Tibetan New Year Party

- 1 units of Bag of flour
- 2 units of Pound of Ground Beef
- 5 units of Clove of Garlic

Next we create the associated view in app/views/grocery_list/show.rhtml:

Download DragAndDropSortableList/app/views/grocery_list/show.rhtml

```
<h2><%= @grocery_list.person.name %>'s Grocery List</h2>
<h3><%= @grocery_list.name %></h3>
<ul id="grocery-list">
  <% @grocery_list.food_items.each do |food_item| %>
  <li id="item_<%= food_item.id %>">
        <%= food_item.quantity %> units of <%= food_item.name %>
  </li>
  <% end %>
</ul>
```

Again, this is nothing too fancy. This is standard Action View read-only material. Do note, though, that we are autogenerating unique element ids for the *<li>* tags. This is necessary when we move on to the sorting code, so don't skip it in this step. We can see what this page looks like by starting our development server and pointing our browser to (assuming the default port) http://localhost:3000/grocery_list/show/*listid*, where *listid* is the id of the GroceryList model object we created in the console.

Now let's make the list sortable. At the end of our show.rhtml, we'll add the following:

Download DragAndDropSortableList/app/views/grocery_list/show.rhtml

```
<%= sortable_element 'grocery-list',
      :url => { :action => "sort", :id => @grocery_list },
      :complete => visual_effect(:highlight, 'grocery-list')
%>
```

This helper generates the JavaScript necessary to turn our unordered list into a dynamic, drag-and-drop sortable form. The first parameter, grocery-list, refers to the ID of the item on the current HTML page that should be transformed into a sortable list. The :url option specifies the elements, such as action and controller, that will make up the URL that will be called when a sorting change is made. We have specified the sort() action of the current controller, appending the current grocery list's ID. Finally, the :complete option sets up a visual effect to take place when the sort() action has finished.

Let's get that sort() action implemented so we can watch this thing in action! In the grocery_list_controller.rb, we'll add a sort() action that looks like this:

Download DragAndDropSortableList/app/controllers/grocery_list_controller.rb

```
def sort
  @grocery_list = GroceryList.find(params[:id])
  @grocery_list.food_items.each do |food_item|
    food_item.position = params['grocery-list'].index(food_item.id.to_s) + 1
    food_item.save
  end
  render :nothing => true
end
```

First we select the grocery list by the supplied ID. Then we iterate through the items on the list and change each item's position to match its index in the grocery-list parameter. The grocery-list parameter is generated automatically by the sortable_element() helper and creates an ordered Array of the list items' IDs. Since our position columns start with 1 and an Array's index starts with 0, we add 1 to the index value before saving the position.

Finally, we explicitly tell Rails that this action should not render anything. Since the visual output of sorting a list is the list itself (which we're already displaying), we let the action complete its work silently. Had we wanted to update the HTML page with the action's results, we could have added the :update option to our sortable_element() call, passing it the ID of the HTML element to populate with our action's results.

If we refresh the grocery list show() page with the sortable_element() addition, we can now drag items up and down the list to change their order both on the page and in the database.

Also See

Chapter 15 of *Agile Web Development with Rails* [TH05] contains a more thorough introduction to acts_as_list().