

Description:

In this lab, you will continue to work on Ruby on Rails, a web application framework based on Ruby. **Your work in this lab will be a continuation of Lab 6. So make sure you have finished Lab 6, before you proceed.** The information you will need is this lab is covered in the Ruby on Rails book.

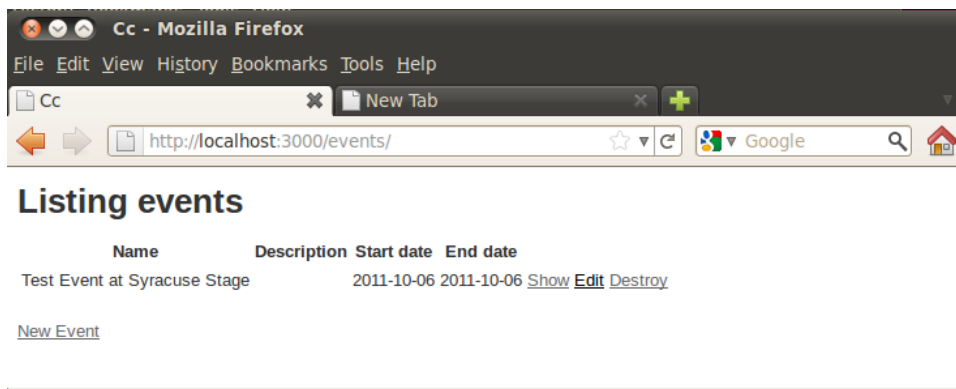
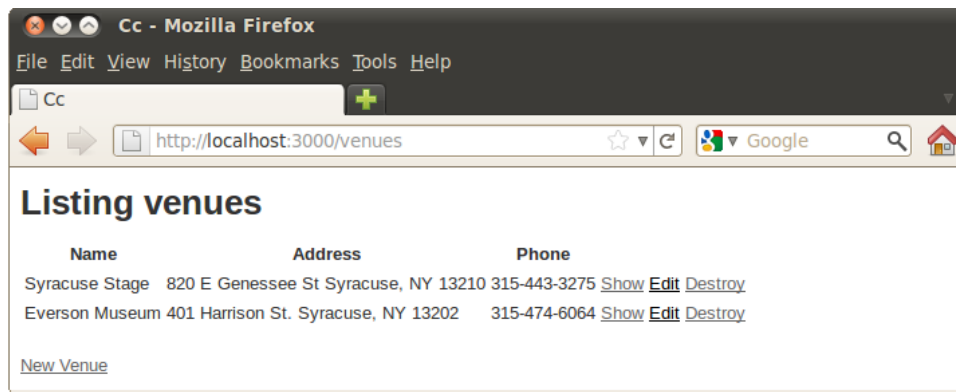
Note: this lab assignment is designed as an individual work. To maximize your learning experience, you are encouraged to work on it independently. You may ask questions to instructors and/or friends (but keep it quiet) and look up resources, but you are not allowed to copy-and-paste any resources other than you created.

You will continue to work on the two models that you previously created -- venues and events.

Lab Instructions:

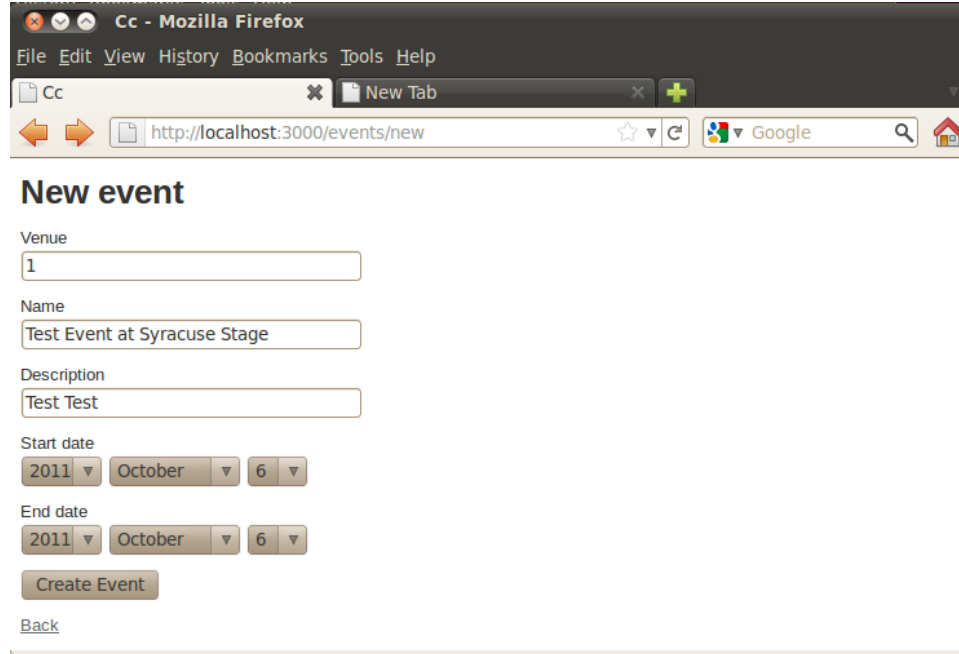
1. Basic Setup

- 1.1. First, login to the Linux system through itellv.ischool.syr.edu, launch a terminal application, and go to the RoR app directory that you worked on last week (i.e. type "cd apps/cc").
- 1.2. If the RoR server is not running, run the server in the app. If you forgot how to do so, see the instructionn in Lab 6.
- 1.3. Launch Firefox and go to <http://localhost:3000/venues> and <http://localhost:3000/events>. You should see screens like below:



2. Modifying the “new” page for events

- 2.1. Now go to `http://localhost:3000/venues/new` on Firefox. Currently, when a user creates a new event, the user has to know the venue id of the venue, which is really lame, so we are going to fix it.



The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://localhost:3000/events/new`. The page title is "New event". The form contains the following fields:

- Venue:
- Name:
- Description:
- Start date:
- End date:

Below the date fields is a "Create Event" button and a "Back" link.

- 2.2. First, open the viewer file for the page, i.e. `app/views/events/new.html.erb` on your favorite text editor. You will see that file only contains three lines:

```
<h1>New event</h1>

<%= render 'form' %>

<%= link_to 'Back', events_path %>
```

- 2.3. Last week, we learned that the codes between `<%=` and `%>` are all string expressions in Ruby. In the case above, the codes are using two methods that are defined by Ruby on Rails: `render` and `link_to`. The names are very much self-explanatory --- `render` renders a file, in this case, `_form.html.erb`, and `link_to` creates generate a hyperlink tag in HTML. (`events_path` is another method that returns a path to the `events/index.html` page.)

- 2.4. Ok, so we actually don't need to modify this file, so close it, and open the `_form.html.erb` file, which contains the actual codes for the form. The file basically looks like this:

```
<% form_for(@event) do |f| %>

  <% if @event.errors.any? %>
    ... error handling here ...
  <% end>

  <div class="field">
    <%= f.label :some_attribute_name %> <br />
    <%= f.text_field :some_attribute_name %>
  </div>

  ... repeat the div for each attribute ...
<% end %>
```

Notice the use of the method `form_for`. It is like the `each` method for arrays or hashes, i.e. it takes a block of code. You don't need to worry too much about the details, but you should at least realize the code block uses a variable `f`, which is associated with the variable `@event`. Now, identify the code that specifying the text form for the `venue_id`, i.e.:

```
<%= f.text_field :venue_id %>
```

You would like to replace this code with something that generates a drop down that lists the venues in the database but how would you do it? Here is the strategy:

- 1) First, obtain all the venues from the database.
- 2) Then, create an array that contains the names of the venues, and ids of the venues. The names are used to be displayed in the dropdown menu, and the ids are used as the return value.
- 3) Using the array, display a dropdown menu.

And here is the code that does that:

```
<% venues = Venue.all %>
<% ids_and_names = venues.map do |v| [v.name, v.id] end %>
<%= f.select :venue_id, ids_and_names, :prompt => "Select a venue..." %>
```

The first line creates a variable, `venues`, which contains all the venues. You might remember `Venue.all` is a method call, which returns all the venues in the database. (In other words, Ruby on Rails executes an SQL call like "select * from venues;" and translates the returned records to an array of ruby objects.) The second line creates another variable, `ids_and_names`, which is an array of pairs of the name of a venue and the id of the venue. The `map` method is similar to the `each` method, except that it returns a new array, each element of which is a returned value from the execution of the code block against each element of the old array. It may sound complicated, so below is a simple example:

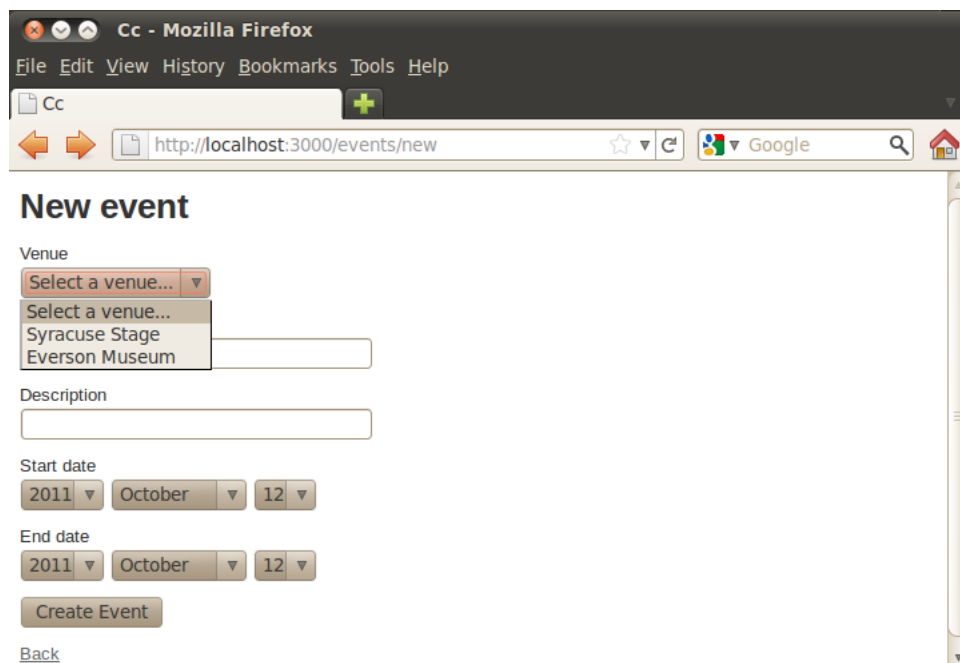
```
[1,2,3,4].map do |x| x * 2 end ➔ [2,4,6,8]
```

The array `ids_and_names` in this case will be something like:

```
[["Syracuse Stage", 1], ["Everson Museum", 2]].
```

And the third line actually generates an HTML code for the dropdown menu.

Once you save the file, go back to Firefox, and reload the page. You should see something like follows:



The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://localhost:3000/events/new`. The page title is "New event". The form contains the following elements:

- Venue:** A dropdown menu with the prompt "Select a venue...". The dropdown is open, showing two options: "Syracuse Stage" and "Everson Museum".
- Description:** A text input field.
- Start date:** A date picker showing "2011" for the year and "October" for the month, with a "12" for the day.
- End date:** A date picker showing "2011" for the year and "October" for the month, with a "12" for the day.
- Create Event:** A button.
- Back:** A link.

3. Adding data entries using Migration

- 3.1. Ok, adding events from the page became a bit less painful, but still, it is chore when you have many events to put. So let's practice adding events using a bulk operation, through migration. First, type in the following commands.

```
script/rails generate migration add_syracuse_stage_events
```

As always, there will be some outputs, showing which files are generated. (Actually, there should be only one file generated from this command.) Since you specified “migration”, this command only generate a migration file, located in the db/migrate directory.

Remember that migration is a feature of Ruby on Rails that manages a sequence of ruby scripts to manage a database. In order to construct a database that fit in with the current app from scratch, Rails needs to run all the scripts in the db/migrate directory in order. When you plan to modify your database (as now) you use the migration command to add a new ruby script which let you specify what to do in order to modify the database (in the up method) or undo the modification (in the down method).

- 3.2. Now, launch your favorite text editor, and open the newly created file. It should be named as YYYYMMDDhhmmss_add_syracuse_stage_events.rb, where YYYYMMDDhhmmss is a time stamp. It should look like below:

```
class AddSyracuseStageEvents < ActiveRecord::Migration
  def self.up
  end
  def self.down
  end
end
```

As you can see, the file defines a class, which extends the ActiveRecord::Migration class. The class has two empty methods: self.up and self.down, which you need to define. For our purpose, you need to define new data entries in the up method, and erase the data entries in the down method.

Note (only if you are curious):

The double columns “::” in “ActiveRecord::Migration” indicates ActiveRecord is a **module**, a group of classes or methods, and Migration is a class in the module. ActiveRecord, as you should know by now, provides the main functionalities of Ruby on Rails.

The prefix ‘self.’ in “self.up” and “self.down” indicates that these methods are **class methods**. Class methods are different from **instance methods** that you learned earlier in the class, in that they are not associated with any instances and can be executed without any instances of the class.

- 3.3. Just to make the migration simpler, let's delete all the events for Syracuse Stage first. This can be done by the following two lines.

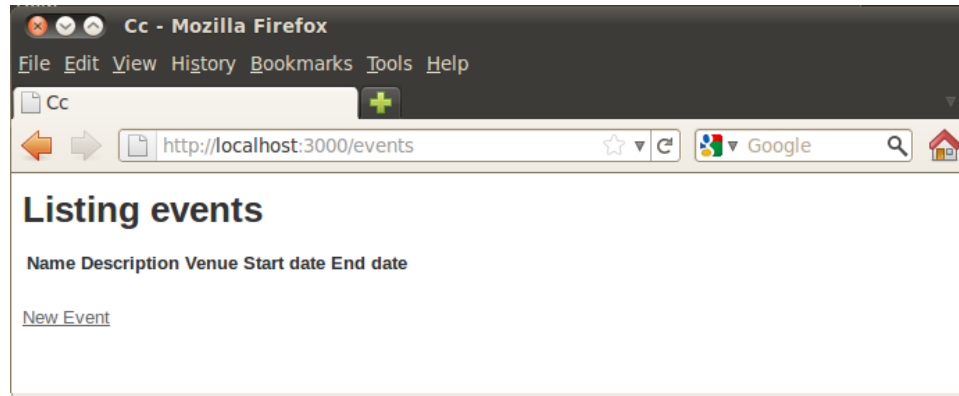
```
venue = Venue.find_by_name("Syracuse Stage")
Event.delete_all(["venue_id = ?", venue.id])
```

The first line is obtaining a venue model object, using the name of the venue. This is a really cool feature of RoR – as soon as you define the model, RoR automatically generates the methods named find_by_XXXX where XXXX can be any attributes. So technically, you use find_by_address or find_by_phone here, too. (But wait, there's more! You can also do find_by_XXXX_and_YYYY where XXXX and YYYY can be any combination of attributes. How cool is that!?)

The second line is deleting all event models of which venue_id is the same as the ID of the Syracuse Stage venue object. The array ["venue_id = ?", venue.id] is kind of like a String expression

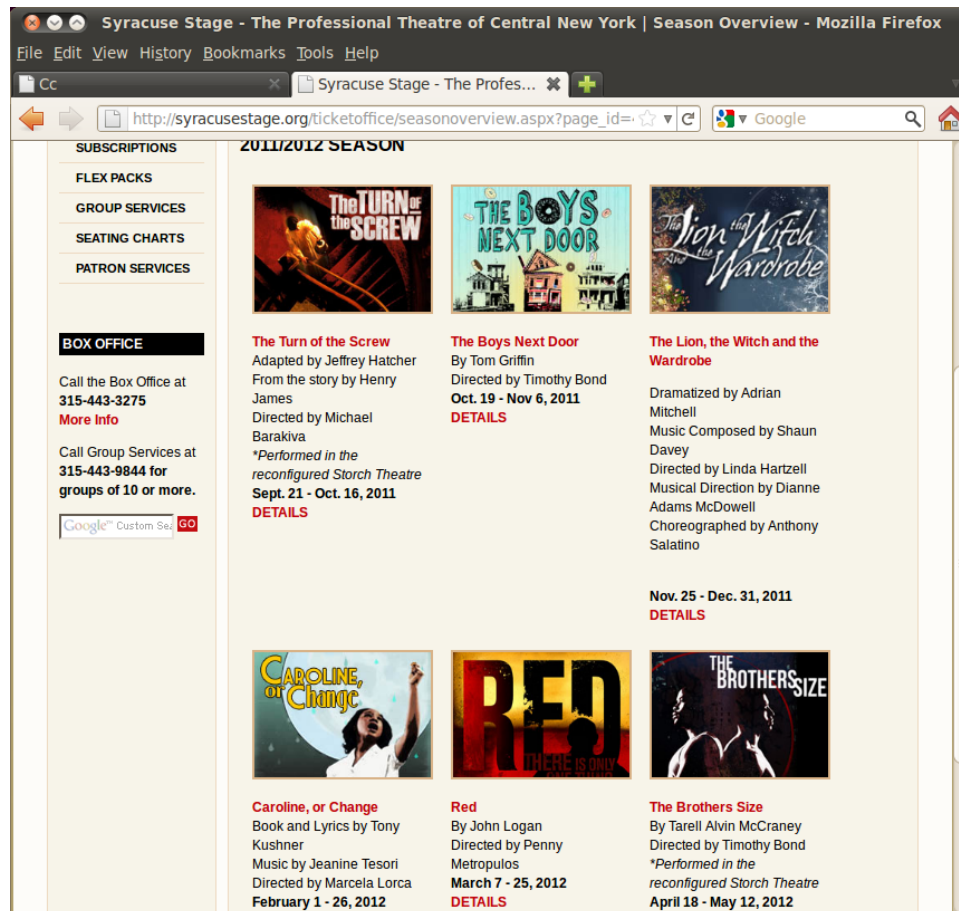
"venue_id = " + venue.id.to_s, but more flexible, because you can insert variable expressions wherever you would like by specifying a space holder ?. (The real reason for using this syntax is for a security reason... it is preventing a very popular hack for SQL (and Perl, and many other script languages) called injection (insertion) attack.

Ok, enough explanations. Just insert these two lines in both self.up and self.down methods, save the file, and run rake db:migration. Now all the events (for Syracuse Stage) should be gone, right?



All the events are gone...

3.4. Now go to <http://syracusestage.org> on FireFox, and checkout up-coming events at Syracuse Stage.



Let's add a few events from their schedule. For example, the following line will create and save the new event for "The Turn of the Screw"

```
Event.create(:venue_id => venue_id,  
            :name => "The Turn of the Screw",
```

```

      :description => "Story by Henry James",
      :start_date => Date.new(2011, 9, 21),
      :end_date => Date.new(2011, 10, 16))

```

The `create` method is like the `new` methods with parameters, except it also saves the object in the database. Notice that the parameters are using hash keys! That is a trick that we learned in the class while ago.

Ok, go ahead and insert the line in the `self.up` method definition. At this point, the entire file should look like the following:

```

class AddSyracuseStageEvents < ActiveRecord::Migration
  def self.up
    venue = Venue.find_by_name("Syracuse Stage")
    Event.delete_all(["venue_id = ?", venue.id])
    Event.create(:venue_id => venue.id,
                 :name => "The Turn of the Screw",
                 :description => "Story by Henry James",
                 :start_date => Date.new(2011, 9, 21),
                 :end_date => Date.new(2011, 10, 16))
  end
  def self.down
    venue = Venue.find_by_name("Syracuse Stage")
    Event.delete_all(["venue_id = ?", venue.id])
  end
end

```

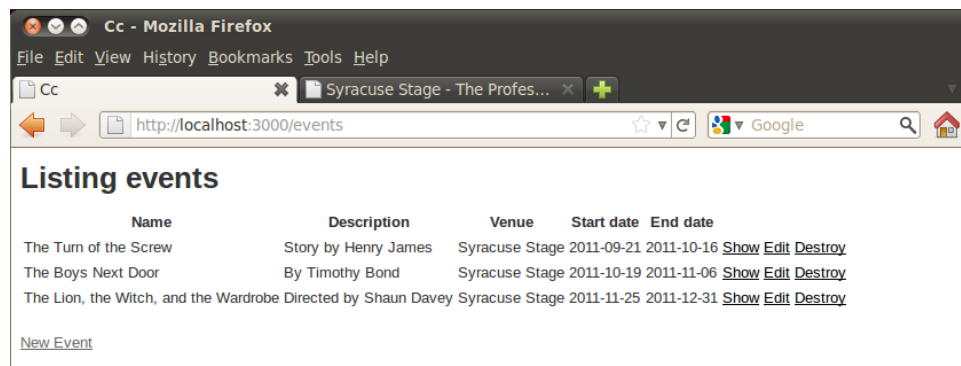
Now, choose a couple of more events from their web site and add the event model creation in the file. Once it's done, save the file.

3.5. Now run the rake command:

```
rake db:migrate:redo STEP=1
```

This is a yet new migration command, which, obviously, redo the migrations for the specified steps. In this case, since the STEP is specified as 1, it will be the one last step, in other words, it will run the `down` and `up` methods in the migration file you are just edited.

3.6. Go back to Firefox and reload <http://localhost:3000/events> You should see more data like below:



Name	Description	Venue	Start date	End date
The Turn of the Screw	Story by Henry James	Syracuse Stage	2011-09-21	2011-10-16 Show Edit Destroy
The Boys Next Door	By Timothy Bond	Syracuse Stage	2011-10-19	2011-11-06 Show Edit Destroy
The Lion, the Witch, and the Wardrobe	Directed by Shaun Davey	Syracuse Stage	2011-11-25	2011-12-31 Show Edit Destroy

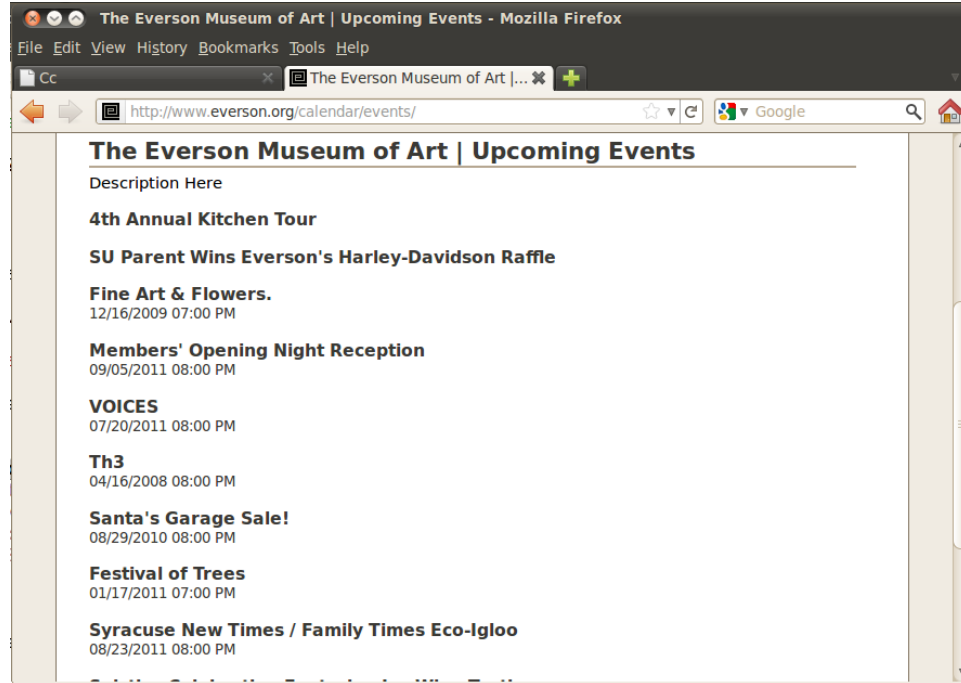
[New Event](#)

4. Creating Everson Museum Events from RSS Feed

Creating events from a Ruby script is good, but seriously, we don't want to type in all the events in the world. Thankfully, some venues' web site generate RSS feed for upcoming events. So we can just parse

their XML. Everson Museum is one of such venues. So let's work on importing their event calendar to our system.

4.1. First check out their RSS feed page <http://www.everson.org/calendar/events>.



Check out the source code of the XML on Firefox. There are a lot of junks in it, but basically, it has a structure like this:

```
<rss>
  <channel>
    <item>
      <title>Event Title</title>
      <startTime>start date and time</startTime>
      <endTime>end date and time</endTime>
    </item>

    <item>
      <title>Event Title</title>
      <startTime>start date and time</startTime>
      <endTime>end date and time</endTime>
    </item>

    ... repeat ...

  </channel>
</rss>
```

Ruby has a standard XML parser called REXML, which we can use to parse the XML. The only tricky part is that startTime and endTime fields are in a unique format, e.g. "Fri Jan 1 2010 00:00:00". There are many way to parse formed date/time strings like this, my suggested strategy here is to split the text string first into an array, and use only the parts we need. For example, "Fri Jan 1 2010 00:00:00".split becomes ["Fri", "Jan", "1", "2010", "00:00:00"]. Now we can use the second, third, and forth elements to create a date object. (The Date class, for some reason, does not accept strings like "Jan", but you can use the Time class to create a Time object, and then convert it into a Date object.)

I have done the most of the work for you, so you can just download the file with the following command:

```
wget http://sanka.syr.edu/files/get_everson_events.rb
```

Now, you can run the script by typing

```
ruby get_everson_events.rb
```

There will be some output from the script, which indicate that some events were read-in. The script, however, does not create events object. (If you want to make sure, check out the event listing on Firefox.) So your task from here, is to create (and save) an Event model object for each event read from the XML. Now, open the file on your text editor.

Identify the line that says: `doc.element.each("rss/channel/item") do |item|`. This is where the XML parser is identifying each RSS feed item (i.e. event). Inside the block, the code is reading in the title, startTime, and endTime fields. Scroll down, and you will see the part that are printing out the read-in event:

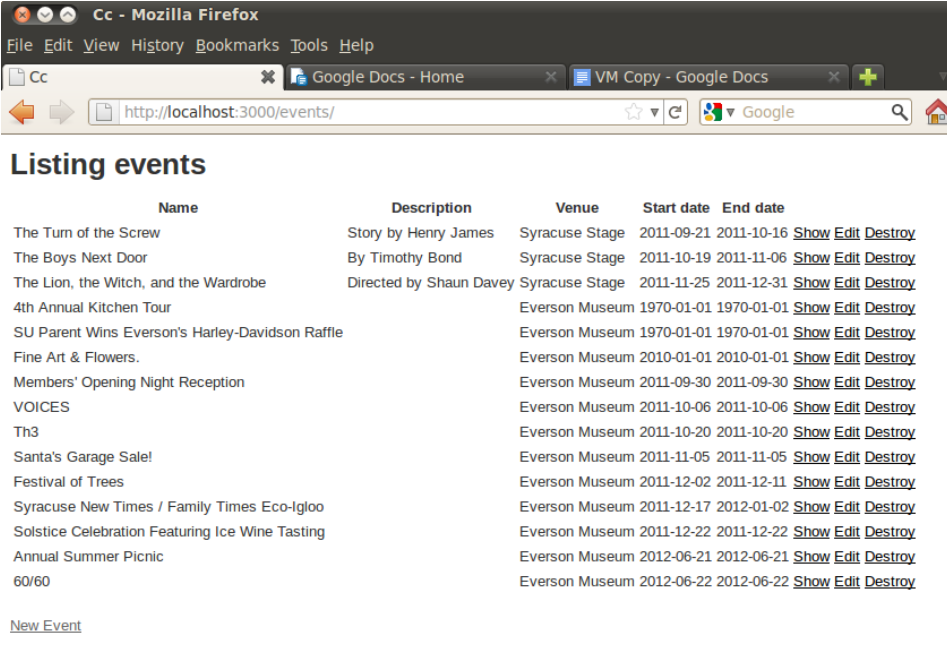
```
print "Reading a new event... \n"
print "  title:      #{title}\n"
print "  start_date:   #{start_date} \n"
print "  end_date:      #{end_date} \n"
```

(By the way using `#{...}`, you can embed string expressions inside a string like this.)

Below that, you need to insert your codes with two steps:

- 1) Find an event object with the same name as the currently read-in event object. Name the variable `event`. (If there is not such event, `nil` should be assigned to the `event` variable.)
- 2) If the `event` variable is `nil`, create a new Event with appropriate parameters.

That's it. Once it's done, save the file and run the script again. Check the event listing on FireFox. You should see quite a few events now.



Name	Description	Venue	Start date	End date
The Turn of the Screw	Story by Henry James	Syracuse Stage	2011-09-21	2011-10-16
The Boys Next Door	By Timothy Bond	Syracuse Stage	2011-10-19	2011-11-06
The Lion, the Witch, and the Wardrobe	Directed by Shaun Davey	Syracuse Stage	2011-11-25	2011-12-31
4th Annual Kitchen Tour		Everson Museum	1970-01-01	1970-01-01
SU Parent Wins Everson's Harley-Davidson Raffle		Everson Museum	1970-01-01	1970-01-01
Fine Art & Flowers.		Everson Museum	2010-01-01	2010-01-01
Members' Opening Night Reception		Everson Museum	2011-09-30	2011-09-30
VOICES		Everson Museum	2011-10-06	2011-10-06
Th3		Everson Museum	2011-10-20	2011-10-20
Santa's Garage Sale!		Everson Museum	2011-11-05	2011-11-05
Festival of Trees		Everson Museum	2011-12-02	2011-12-11
Syracuse New Times / Family Times Eco-Igloo		Everson Museum	2011-12-17	2012-01-02
Solstice Celebration Featuring Ice Wine Tasting		Everson Museum	2011-12-22	2011-12-22
Annual Summer Picnic		Everson Museum	2012-06-21	2012-06-21
60/60		Everson Museum	2012-06-22	2012-06-22

[New Event](#)

5. Adding Events from Other Venues

Now that you learned how to import Events from RSS feed. Let's repeat the process for another venue. You can choose one of the venues from the Connective Corridor's Web site <http://connectivecorridor.syr.edu/corridor-venues/>.

Here are the basic steps you can take:

- Find a venue that supports RSS feed from the list.
- Check out the RSS feed format. Make sure they are usable for creating Event object in your system. Identify which XML fields correspond to name, description, start_date, and end_date attribute of the Event model. (Not all of them have to be supplied.)
- Create a venue on your system. (You can do it either manually or by using a script.)
- Copy the get_everson_events.rb script and modify it, so that it would work with the new venue.
- Run the script (obviously).

That's it!

So far, I identified the following two venues support RSS feed other than Everson. You can work in with either one of them, but extra 2 points will be given as a bonus, if you choose a different venue:

- Armory Square
- Westcott Theater

6. Submission

1. On your terminal window, go to the `apps` directory (i.e. if you are in the `cc` directory, type "`cd ..`").
2. Type in the following command:

```
tar cvfz cc.tgz cc
```

You should see a bunch of output and a new file `cc.tgz` created. (This is a compressed archive of the `cc` folder.)

3. Go to: <http://sanka.syr.edu/swt/> and submit the file.