Flash Video Tutorial with Rails, ffmpeg, FlowPlayer, and attachment_fu

Quick Tutorial today to get a simple Video Model setup and going with Flowplayer today. We want to be able to upload videos and convert them to flash .flv files. For this I'll be using ffmpeg. Other plugins in this tutorial include acts_as_state_machine and Rick Olsen's attachment_fu. Lets get started!

Note: You can find the source code for this app on github at http://github.com/balgarath/video-app/tree/master. EDIT: I've been working on this, so if you want to see the code specific to the tutorial, click on the Tutorial branch on github.

1. Setup

First, lets create an empty app

```
rails videoapp -d mysql
```

Do all your basic setup...(edit database.yml, remove public/index, ...). In routes.rb, add this line:

```
  map.root :controller => "videos"
map.resources :videos
```

The first line maps '/' to the videos controller, and the next establishes RESTful routes for videos.

Now we need to figure out what flash video player we want to use. For this tutorial, I will be using FlowPlayer, an open-source flash video player. You will need to download flowplayer.zip from http://flowplayer.org/download/index.html. Unzip it to /public. Grab the /example/flowplayer-3.0.2.min.js file and put it in /public/javascripts/flowplayer.js. Put example/style.css in /public/stylesheets.

Next we need to install attachment_fu to handle file downloads.

```
./script/plugin install http://svn.techno-weenie.net/projects/plugins/attachment_fu/
```

And you will need ffmpeg for converting uploaded file to .swf. This works for me in Ubuntu...

```
sudo apt-get install ffmpeg
```

And last, lets install the acts_as_state_machine plugin. We will be using it to keep track of the current state of conversion for the video:

```
 ./script/plugin install http://elitists.textdriven.com/svn/plugins/acts_as_state_machine/trunk/
```

2. Database/Model Setup

Now we are ready to set up the model. First, run this command to generate the files for you:

```
  ./script/generate model Video
```

This last line will also generate a create_video migration in db/migrate/.

Open up that file and put this in it:

```
class CreateVideos < ActiveRecord::Migration
def self.up
create_table :videos do |t|
t.string :content_type
t.integer :size
t.string :filename
t.string :title
t.string :description
t.string :state
t.timestamps
end
end

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

Content_type, size, and filename are used by the attachment_fu plugin. The state field will be used by the act_as_state_machine plugin to keep track of video converting.

Lets move on to the model; open up /app/model/video.rb and add this into it:

```
has_attachment :content_type => :video,
:storage => :file_system,
:max_size => 300.megabytes

#turn off attachment_fu's auto file renaming
#when you change the value of the filename field
def rename_file
true
end
```

This is for the attachment_fu plugin. I came across a feature of attachment_fu: when you change the filename of an attachment, the old file automatically gets moved to whatever the new filename is. Since I am creating a new file and changing the filename attribute of the video the reflect that, I don't need this on...so I just overrode the method in the Model.

Since we will be using the acts_as_state_machine plugin to keep track of the state of conversion for videos, lets go ahead and add in the states we will be using. Add this to video.rb:

```
#acts as state machine plugin
acts_as_state_machine :initial => :pending
state :pending
state :converting
state :converted, :enter => :set_new_filename
state :error

event :convert do
transitions :from => :pending, :to => :converting
end

event :converted do
transitions :from => :converting, :to => :converted
```

```
end

event :failure do
transitions :from => :converting, :to => :error
end
```

Note: I got some of this part from Jim Neath's tutorial.

Now we can use @video.convert!, @video.converted!, @video.failed! to change the state of a particular Video. The last code we need to add to the model is this:

```
# This method is called from the controller and takes care of the converting
def convert
self.convert!
success = system(convert_command)
if success && $?.exitstatus == 0
self.converted!
else
self.failure!
end
end

protected

def convert_command

#construct new file extension
flv =  "." + id.to_s + ".flv"

#build the command to execute ffmpeg
command = <<-end_command
ffmpeg -i #{ RAILS_ROOT + '/public' + public_filename }  -ar 22050 -ab 32 -s 480x360 -vcodec flv -r 25

end_command

logger.debug "Converting video...command: " + command
command
end

# This updates the stored filename with the new flash video file
def set_new_filename
update_attribute(:filename, "#{filename}.#{id}.flv")
update_attribute(:content_type, "application/x-flash-video")
end
```

A lot of stuff going on here. Convert will be called from the create action. When called, it sets the state of the video to 'convert' and runs ffmpeg to convert the file to flash(.flv). It will then mark the file as either 'converted' or 'failed'.

Now that that is all done, we can create our database and run the migrations:

```
rake db:create
rake db:migrate
```

3. Controller/Views

Now we can generate our controller, model, and view files:

```
 ./script/generate controller videos index show new
```

Open up /app/controllers/videos_contoller.rb and put in this code:

```
class VideosController < ApplicationController

def index
@videos = Video.find :all
end

def new
@video = Video.new
end

def create
@video = Video.new(params[:video])
if @video.save
@video.convert
flash[:notice] = 'Video has been uploaded'
redirect_to :action => 'index'
else
render :action => 'new'
end
end

def show
@video = Video.find(params[:id])
end

def delete
@video = Video.find(params[:id])
@video.destroy
redirect_to :action => 'index'
end
end
```

In the create method, notice that if the video save is true, @video.convert gets called, which convert the video to .flv.

Views

Create the file /app/views/layouts/application.html.erb and put this in it:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>
<title>Video Tutorial</title>
<%= stylesheet_link_tag 'style' %>
<%= javascript_include_tag 'flowplayer' %>
</head>

<body>
<h1>Video Tutorial</h1>

<% flash.each do |key,value| %>

<div id="flash" class="flash_<%= key %>" >
<span class="message"><%= value %></span>
```

```
</div>

<% end -%>

<%= yield :layout %>

</body>

</html>
```

Now for our index view (/app/views/videos/index.html.erb)

```
<h2>Videos</h2><br />

<%= link_to 'New Video', new_video_url %><br />
<% for video in @videos do %>
<p><%= link_to video.title, video %></p>
<% end %>
```

/app/views/videos/new.html.erb:

```
<h2>New Video</h2><br />

<% form_for(@video, :html => { :multipart => true }) do |f| %>
<%= f.error_messages %>
<table>
<tr>
<td><%= f.label :title %>: </td><td><%= f.text_field :title %></td>
</tr>
<tr>
<td><%= f.label :description %>: </td><td><%= f.text_area :description %></td>
</tr>
<tr>
<td><%= f.label :video %>: </td><td><%= f.file_field :uploaded_data %></td>
</tr>
<tr><td><%= f.submit 'Submit' %> - <%= link_to 'Back', videos_path %></td></tr>
</table>

<% end %>
```

This is the upload form. Notice I used f.file_field :uploaded_data...this is for attachment_fu to work. Next is /app/views/videos/show.html.erb:

```
<h1><%= @video.title %></h1>
<p><%= @video.description %></p>

<a
href="<%= @video.public_filename %>"
style="display:block;width:400px;height:300px"
id="player">
</a>

<!-- this will install flowplayer inside previous A- tag. -->
<script>
flowplayer("player", "/flowplayer/flowplayer-3.0.3.swf");
</script>
```

And that should do it. ./script/server and try uploading a movie file and see if it works. Also, you could probably mess around some with the call to ffmpeg and increase video quality. There are some good posts if you search for 'ffmpeg' over at the FlowPlayer Forums, and if you purchase a commercial license for the player, you can remove the advertising and get some new features, as well as support. Thanks for reading!

Update:

For a quick way to put the video conversion on a background process, I was able to use Tom Anderson's Spawn plugin. Note that this wouldn't be a very efficient way to scale it if you expect to have a lot of users uploading at the same time, as it forks another process to handle the conversion. This does work well if there aren't a bunch of videos getting uploaded at the same time. If you do need to scale the uploading, I recommend using Ap4r(Asynchronous Processing for Ruby). Here we go!

First, lets install the Spawn plugin from github:

```
./script/plugin install git://github.com/tra/spawn.git
```

And next, change the convert method in Video.rb to this:

```
# This method is called from the controller and takes care of the converting
def convert
self.convert!

#spawn a new thread to handle conversion
spawn do

success = system(convert_command)
logger.debug 'Converting File: ' + success.to_s
if success && $?.exitstatus == 0
self.converted!
else
self.failure!
end
end #spawn
end
```

And thats it! Now when you upload a video, you don't have to wait for the server to convert the video file over to flash.