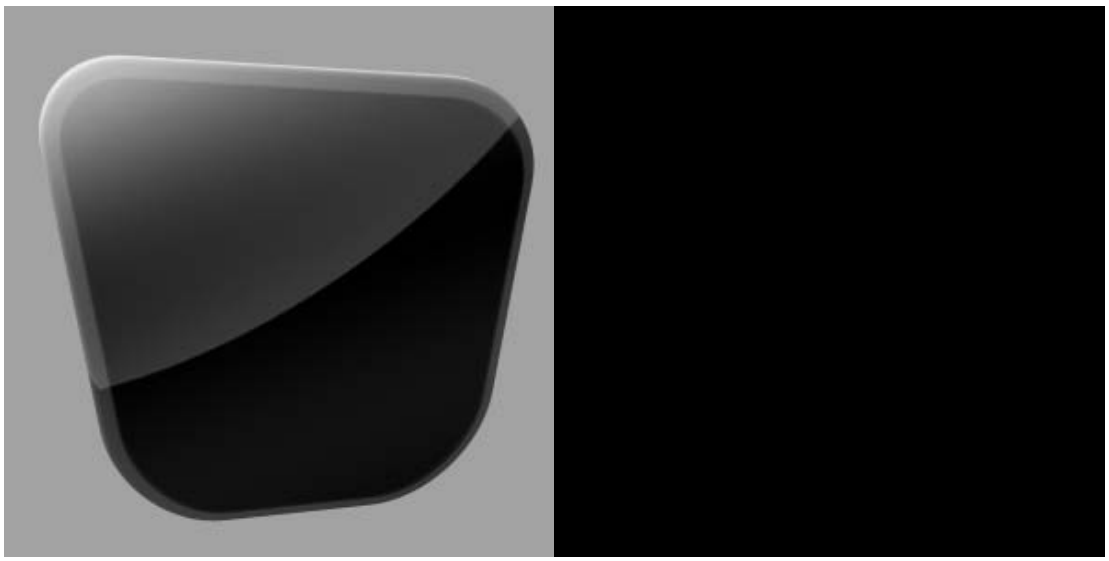
UploadColumn

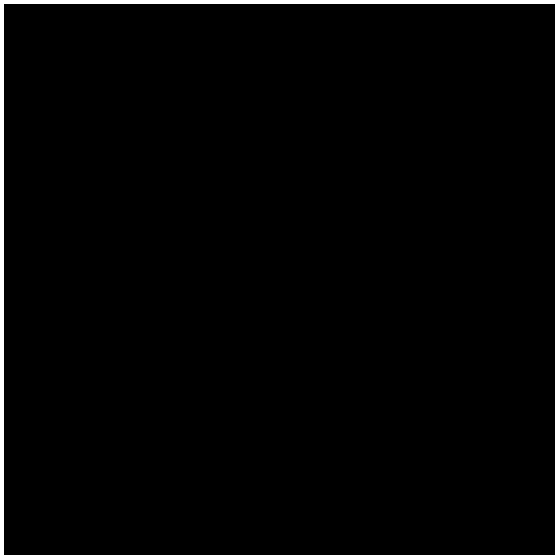
This version of the tutorial is written for UploadColumn 0.2.X and some parts will not work with newer versions!

Working with Images

This page will follow more of a tutorial style. I am assuming that you have some knowledge of Photoshop, we'll also use Illustrator, but we won't do anything hard. Let's do something tricky: Our webdesigner has asked as for rounded corners (they always do). sounds okay, but he wants them on the users avatars (pictures). We could solve this by using transparent PNGs, but that leads to some fairly arcane markup and CSS. Our graphic designer also wants the user to be able to decide the color of her (for the purpose of this article all users are female) page, and the rounded corners have to work with that. Now this as far as I know can't be done in CSS. We want our result to look something like this:

There are several elements to this image, this is not a Photoshop tutorial, so I've provided the needed images for you, the part we're interested in is merging these together. Have a look at the images:





Note that I have added a chequered background, so you can easily see which areas of the images are transparent, that background is not part of the images. The first image is the blingy stuff, the highlights and gradients and such, the frame (passe-par-tout?) is unimportant (even undesired) but it was easier to export with the frame from Photoshop. This image we'll simply add on top of our users' images.

The second image is is our transparency template, this is the easiets and cleanest way I know for changing the background color. We create an image with a solid background fill of the desired color, then we copy the alpha channel from our template image to the fill image.

The third image is our shadow, this one we'll add on top of the solid color fill image, right before we copy the alpha channel

To summarise:

1. Scale the image
2. Compose the various blingy things, shadows and gradient etc. over the image
3. Create a blank image and fill it with the background color
4. Compose the drop shadow image over
5. Copy the alpha channel from our alpha channel template.
6. Merge the two images

Down to business

Generate a rails app, edit your database.yml and generate a model for user (you can do this by yourself I hope, otherwise, read some rails tutorials!). Rails will automatically have created a migration for you, open `db/migrate/001_create_users.rb` and change self.up to:

```
def self.up
    create_table :users do |t|
        t.column :name, :string
        t.column :color, :string
        t.column :avatar, :string
    end
end
```

Execute the migration by typing `rake db:migrate` at the top directory of your rails app. Now generate a scaffold for your model. You have now created a table called `users` in your database, with the four columns id, name, color, string. You can do this in your favorite database frontend as well, but rails' migrations is a great feature that's well worth learning. Install UploadColumn by typing:

```
script/plugin install svn://rubyforge.org/var/svn/uploadcolumn/tags/stable
```

Open app/models/user.rb and add the image_column instruction:

```
image_column :avatar, :versions => { :thumb => "50x50", :fancy => "c300x300" }
```

Open app/views/users/_form.rhtml and add fields for name, color and avatar, make avatar an upload_column_field.

```
<p><label for="user_name">Name</label><br/>
<%= text_field 'user', 'name' %></p>

<p><label for="user_color">Color</label><br/>
<%= text_field 'user', 'color' %></p>

<p><label for="user_avatar">Avatar</label><br/>
<%= upload_column_field 'user', 'avatar' %></p>
```

Open app/views/users/new.rhtml and change the `start_form_tag` instruction to `upload_form_tag`. Do the same for edit.rhtml.

We're cropping the version `:fancy` to exactly 300x300 pixels, whereas `:thumb` will have the original aspect ratio and not be bigger than 50x50. This is done by prepending a 'c' to the size string.

Start your server and upload an image to test that it works.

Download the images I've provided and place them in the public/images folder of your rails app.

For now we'll do the composing at runtime, instead of when the image is saved, because that way we can see the results without having to upload a new image every time. Add an action called 'avatar' to your users controller, fetch the user from the passed ID and render the fancy version of the uploaded image:

```
def avatar
    @user = User.find(params[:id])
    render_image( @user.avatar.fancy ) do |img|
        img
    end
end
```

Point your browser at http://localhost:3000/users/avatar/1. You should see the image you uploaded earlier, make sure it looks okay and is the right size. Now let's start manipulating it!

First we'll add the bling stuff, we'll need to load the image, and since we're going to load several images, we'll add a method call for that, make it protected so it can't be called as an action (we don't want people to be able to do exmample.com/users/load_image).

```
protected

def load_image(image)
    path = File.join(RAILS_ROOT, 'public', 'images', image)
    return ::Magick::Image::read(path).first
end
```

First we concatenate the path, we use `File.join` since we can't be sure what systems we'll need this to work on (remember, Windows uses backslashes, the civilized world uses forwardslashes). Then we load (and return) that image using RMagick.

Now let's get to it:

```
def avatar
    @user = User.find(params[:id])
    render_image( @user.avatar.fancy ) do |img|
        bling = load_image('bling.png')
        img.composite!(bling, 0, 0, ::Magick::OverCompositeOp)
    end
```

```
end
```

First we load in the bling image, then we composite it over. To `composite!` we'll pass the image, the coordinates, in our case the top left corner and the composite operator OverCompositeOp. Your image should look like this:

Now let's create the solid fill image:

```
frame = ::Magick::Image::new(300,300)
frame = frame.color_floodfill(5,5,"#334455")
```

For now we'll use just some random color, it doesn't really matter which. Note that we created an image that is 300 by 300 pixels large. `color_floodfill` demands that you specify coordinates where the fill begins, this doesn't really matter here, I used 5 and 5.

```
shadow = load_image('shadow.png')
frame.composite!(shadow, 0, 0, ::Magick::OverCompositeOp)
```

We add the shadow to our frame. If you add:

```
frame.format = 'PNG'
frame
```

you can output the frame (RMagick won't render the image unless you specify a format), it should look something like this:

Your avatar action should now look like this:

```
def avatar
    @user = User.find(params[:id])
    render_image( @user.avatar.fancy, 'image/png') do |img|
        img.format = 'PNG'

        # Compose bling over img
        bling = load_image('bling.png')
        img.composite!(bling, 0, 0, ::Magick::OverCompositeOp)

        # Create a solid fill image
        frame = ::Magick::Image::new(300,300)
        frame = frame.color_floodfill(5,5,"#334455")

        # Load the shadow image
        shadow = load_image('shadow.png')

        # composite them together
        frame.composite!(shadow, 0, 0, ::Magick::OverCompositeOp)
        frame.format = 'PNG'
        rame
    end
end
```

We'll load our transparency template image and copy the alphachannel over:

```
def avatar
    @user = User.find(params[:id])
    render_image( @user.avatar.fancy, 'image/png') do |img|
        img.format = 'PNG'

        # Compose bling over img
        bling = load_image('bling.png')
        img.composite!(bling, 0, 0, ::Magick::OverCompositeOp)

        # Create a solid fill image
        frame = ::Magick::Image::new(300,300)
        frame = frame.color_floodfill(5,5,"#334455")

        # Load the shadow image
        shadow = load_image('shadow.png')

        # composite them together
        frame.composite!(shadow, 0, 0, ::Magick::OverCompositeOp)

        # Copy over the alpha channel from our template
        alpha = load_image('alpha.png')
        frame.composite!(alpha, 0,0, ::Magick::CopyOpacityCompositeOp)

        frame.format = 'PNG'
        frame
    end
end
```

The image should look like this:

This is our frame, we simply composite this over our image and we are presented with an image that looks pretty much like we wanted!

```
def avatar
    @user = User.find(params[:id])
    render_image( @user.avatar.fancy, 'image/png') do |img|
        img.format = 'PNG'

        # Compose bling over img
        bling = load_image('bling.png')
        img.composite!(bling, 0, 0, ::Magick::OverCompositeOp)

        # Create a solid fill image
        frame = ::Magick::Image::new(300,300)
        frame = frame.color_floodfill(5,5,"#334455")

        # Load the shadow image
        shadow = load_image('shadow.png')

        # composite them together
        frame.composite!(shadow, 0, 0, ::Magick::OverCompositeOp)

        # Copy over the alpha channel from our template
        alpha = load_image('alpha.png')
        frame.composite!(alpha, 0,0, ::Magick::CopyOpacityCompositeOp)

        img.composite!(frame, 0,0, ::Magick::OverCompositeOp)
    end
end
```

Can you spot which one is Photoshop and which one is RMagick?

Along the bottom edge, the RMagick image (left) has worse quality than the Photoshop equivalent, if we had used a better image for the blingy stuff we might have been able to avoid that.

We're not done yet! At the moment we are rendering the image at runtime, this works, but it's quite inefficient, we should do all the compositing directly after we save the image.

Already we (or at least I, because I wrote the thing) can foresee a problem: UploadColumn _after_assign callbacks are called when the file is assigned to the attribute in our model, however at that stage we cannot be certain of the background color, therefore we need to find another way.

```
@user = User.new
@user.avatar = params[:user][:avatar] # we don't know @user.color yet!
@user.color = params[:user][:color]
```

Even if we do mass-assignment (`@user = User.new(params[:user])`) we can't be sure which attribute will be assigned first. We can work around this in two ways: make sure avatar is always assigned after color, or move the image compositing into an after_save callback. We'll go for the after_save version because that way we can even make sure that the image always has the right background color, even if the user edits it; after_assign callbacks only get called when a new image is uploaded.

At this point it might be wise to add `image_tag(@user.avatar.fancy.url)` somewhere in your code so you can actually see what you're doing.

Add an after_save callback to your model:

```
after_save :fancify

def fancify
end
```

We'll need to check if there actually is an image, because avatar might be blank, then we'll manipulate the image exactly like we did before:

```
after_save :fancify

def fancify
    if self.avatar
        self.avatar.fancy.process! do |img|

            # Compose bling over img
            bling = load_image('bling.png')
            img.composite!(bling, 0, 0, ::Magick::OverCompositeOp)

            # Create a solid fill image
            frame = ::Magick::Image::new(300,300)
```

```
            frame = frame.color_floodfill(5,5,"#334455")

            # Load the shadow image
            shadow = load_image('shadow.png')

            # composite them together
            frame.composite!(shadow, 0, 0, ::Magick::OverCompositeOp)

            # Copy over the alpha channel from our template
            alpha = load_image('alpha.png')
            frame.composite!(alpha, 0,0, ::Magick::CopyOpacityCompositeOp)

            img.composite!(frame, 0,0, ::Magick::OverCompositeOp)
        end
    end
end
```

protected

def load_image(image) path = File.join(RAILS_ROOT, 'public', 'images', image) ::Magick::Image::read(path).first end

This should work, but if we edit our image, we'll do this transformation multiple times, that's not what we want! To fix this we'll add another version to our image_column declaration, we'll call it `primer`.

```
image_column :avatar, :versions => { :thumb => "50x50", :fancy => "c300x300",
```

We'll add an after_assign callback, where we'll add the bling image to primer:

```
def avatar_after_assign
    self.avatar.primer.process! do |img|
        # Compose bling over img
        bling = load_image('bling.png')
        img.composite!(bling, 0, 0, ::Magick::OverCompositeOp)
    end
end
```

Now change fancify to:

```
def fancify
    if self.avatar
        self.avatar.fancy.process! do |img|
            # Load primer
            img = ::Magick::Image::read( self.avatar.primer.path ).first

            # Create a solid fill image
            frame = ::Magick::Image::new(300,300)
            ...
        end
    end
end
```

And for the icing on the cake, use the actual color (In this example we expect the color to be given without the pound sign (#)):

```
# Create a solid fill image
frame = ::Magick::Image::new(300,300)
frame = frame.color_floodfill(5,5,"#" + self.color)
```

Now hopefully everything should be working. Don't forget to remove the avatar action and the load_image method from your controller.

In a real world application you'll probably want to give the user the choice between a few select colors, your average Joe does not know hex codes :P At this point, your model should look like this:

```
class User < ActiveRecord::Base
    image_column :avatar, :versions => { :thumb => "50x50", :fancy => "c300x300

    def avatar_after_assign
        self.avatar.primer.process! do |img|
            # Compose bling over img
            bling = load_image('bling.png')
            img.composite!(bling, 0, 0, ::Magick::OverCompositeOp)
        end
    end

    after_save :fancify

    def fancify
        if self.avatar
            self.avatar.fancy.process! do |img|
                # Load primer
                img = ::Magick::Image::read( self.avatar.primer.path ).first

                # Create a solid fill image
                frame = ::Magick::Image::new(300,300)
                frame = frame.color_floodfill(5,5,'#' + self.color)

                # Load the shadow image
                shadow = load_image('shadow.png')

                # composite them together
                frame.composite!(shadow, 0, 0, ::Magick::OverCompositeOp)

                # Copy over the alpha channel from our template
                alpha = load_image('alpha.png')
                frame.composite!(alpha, 0,0, ::Magick::CopyOpacityCompositeOp)

                img.composite!(frame, 0,0, ::Magick::OverCompositeOp)
            end
        end
    end

    protected

    def load_image(image)
        path = File.join(RAILS_ROOT, 'public', 'images', image)
        ::Magick::Image::read(path).first
    end
end
```

Horrible JPEG compression

The image you upload is most likely a JPEG
image, which probably means, the compression is really, really awful. JPEG just doesn't play nice with solid fills. PNG
would be a better choice, in UploadColumn 2.1 you'll have the option of forcing all your images to a specific format (such as PNG). If you are using 2.1 or higher (as of writing 2.1 is not released yet), then you can pass `:force_format => :png` to your image_column declararion.

You can get the finished app here. If you have sqlite3 you should be able to run it as-is, then just point your browser to http://localhost:3000/users.