Extracted from:

Rails for PHP Developers

we've created an uploads directory to hold all the file uploads in our application.

PHP

Download php_to_rails/php/files/file_upload.php

```
// destination for file
$destDir = dirname(__FILE__)."/uploads/";
$dest    = $destDir.basename($_FILES['uploaded_file']['name']);

if (move_uploaded_file($_FILES['uploaded_file']['tmp_name'], $dest)) {
    echo "File uploaded successfully.";
}
```

Ruby

Download php_to_rails/ruby/demo_1/app/models/document.rb

```
before_save :write_file_upload

def write_file_upload
  dest = "#{RAILS_ROOT}/uploads/#{self.filename}"
  File.open(dest, 'w') {|f| f << @contents }
end
```

You'll obviously want to perform some validation and error checking for your file uploads just as you would in PHP. You would typically check that the file size isn't zero and that the uploaded file doesn't already exist on disk. Rick Olson has written a useful Rails plug-in that deals with a lot of the issues you may run into while dealing with file uploads. This plug-in is named attachment_fu and can be found in Rick's SVN Repository.[1]

13.4 $_SERVER

Most of the common environment variables you would get through the $_SERVER superglobal array or getenv() function in PHP are set as methods on the request object in Rails. As shown in Figure 13.1, on the following page, we can access these by referencing these methods from within a controller action.

13.5 Cookies

Setting cookies in Rails is done by assigning a value to the cookies hash within a controller action. We can also assign a hash of parameters to the cookie if we need to specify the expiration date or path constraint.

1. http://svn.techno-weenie.net/projects/plugins/attachment_fu/

PHP	Rails
<?php	def my_action
$_SERVER['REQUEST_METHOD'];	request.method
$_SERVER['REQUEST_METHOD'] == 'GET';	request.get?
$_SERVER['REQUEST_METHOD'] == 'POST';	request.post?
$_SERVER['REQUEST_METHOD'] == 'PUT';	request.put?
$_SERVER['REQUEST_METHOD'] == 'DELETE';	request.delete?
$_SERVER['HTTP_ACCEPT'];	request.accepts
$_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest';	request.xhr?
$_SERVER['REMOTE_ADDR'];	request.remote_ip
$_SERVER['SERVER_SOFTWARE'];	request.server_software
!empty($_SERVER['HTTPS']);	request.ssl?
$_SERVER["HTTP_HOST"];	request.host
$_SERVER['SERVER_PORT'];	request.port
$_SERVER['REQUEST_URI'];	request.request_uri
	end

Figure 13.1: Server variables

PHP

Download php_to_rails/php/cookies/set_cookies.php

```
// expire at the finish of the current session
setcookie('tabState', 'open');

// set additional info for the cookie
setcookie("tabState", 'open', time()+3600*24*14, "/~foo/");
```

Ruby

Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  # expire at the finish of the current session
  cookies[:tab_state] = 'open'

  # set additional info for the cookie
  cookies[:tab_state] = { :value   => 'open',
                          :expires => 14.days.from_now,
                          :path    => "/~foo/" }
end
```

We can retrieve cookies within a Rails controller by simply accessing the value for the cookie from the cookies hash. Remember that this method is not a superglobal such as the $_COOKIE array in PHP and is available only when working in an action or view.

PHP Download php_to_rails/php/cookies/get_cookies.php

```
$state = isset($_COOKIE['tabState']) ? $_COOKIE['tabState'] : null;
```

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  state = cookies[:tab_state]
end
```

We delete cookies in PHP by setting an expiration date that has already passed. In Rails, we delete a cookie using the delete method to our cookies proxy object. Simply call this method with the name of the cookie you want to wipe out.

PHP Download php_to_rails/php/cookies/delete_cookies.php

```
// one hour ago
setcookie("tabState", "", time() - 3600);
```

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  cookies.delete(:tab_state)
end
```

13.6 Sessions

Session data is set within controller methods by assigning values to the session hash. There is no need for any equivalent of PHP's session_start function.

PHP Download php_to_rails/php/sessions/set_session.php

```
session_start();

$_SESSION['user'] = $user->id;
```

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  session[:user] = @user.id
end
```

We retrieve session data in Rails by accessing values of the session hash by key name. This method is not a superglobal such as the $_SESSION array in PHP and is available only when working in an action or view.

PHP Download php_to_rails/php/sessions/get_session.php

```
session_start();

$userId = isset($_SESSION['user']) ? $_SESSION['user'] : null;
```

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```ruby
def my_action
  user_id = session[:user]
end
```

We can clear all existing session data using the reset_session method, which works similarly to PHP's session_destroy function.

PHP Download php_to_rails/php/sessions/reset_session.php

```php
session_destroy();
```

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```ruby
def my_action
  reset_session
end
```

There are various session storage options in Rails that can be changed to suit your needs. The default session storage mechanism uses cookies and is suitable for most needs. However, in some cases, you may need to store more session data than allowed in a cookie (4KB). You might also at times want to store sensitive information that you would rather not have stored in a cookie. In these scenarios, you may want to use ActiveRecord to store your sessions in the database. Turning on :active_record_store can be done by uncommenting the session_store assignment in the initializer block in config/environment.rb.

Ruby Download php_to_rails/ruby/demo_1/config/environment.rb

```ruby
config.action_controller.session_store = :active_record_store
```

If we want to use Rails' built-in cross-site request forgery protection, we need to perform an additional step when switching the session store. Any session store other than the default cookies storage requires us to provide a :secret token to the protect_from_forgery method in app/controllers/application.rb. This token is already generated in your source code and just needs to be commented out to work with our active record session storage.

Ruby Download php_to_rails/ruby/demo_1/app/controllers/application.rb

```ruby
protect_from_forgery :secret => 'ef992b27ee422f2e5b5e44bab9e6f7e0'
```

Once we've done this, we need to create the sessions migration to create the database table needed to store our data. We can do this using a Rake task bundled with Rails.

From your application's root directory, run the following:

```
demo> rake db:sessions:create
(in /Users/derek/work/demo)
exists  db/migrate
create  db/migrate/003_create_sessions.rb
```

Now we can use the new session migration to add this table to our database:

```
demo> rake db:migrate
(in /Users/derek/work/demo)

== 3 CreateSessions: migrating ================================================
-- create_table(:sessions)
   -> 0.0503s
-- add_index(:sessions, :session_id)
   -> 0.0086s
-- add_index(:sessions, :updated_at)
   -> 0.0559s
== 3 CreateSessions: migrated (0.1157s) =======================================
```

Once we've restarted the server, sessions will now be stored in the sessions table instead of the default cookie storage. If we ever want to clear our active record session data, there is another Rake task to handle this.

```
demo> rake db:sessions:clear
```

13.7 Headers and Redirection

We can send arbitrary headers in a controller method by assigning header values on the response object. This works similarly to PHP's header function.

PHP

Download php_to_rails/php/headers/headers.php

```
header('Cache-Control: no-cache, must-revalidate');
header('Content-Type: application/pdf');
```

Ruby

Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  response.headers['Cache-Control'] = 'no-cache, must-revalidate'
  response.headers['Content-Type']  = 'application/pdf'
end
```

Rails provides a method in our controllers to set proper redirect headers in our application. The redirect_to method uses a hash of parameters that compose the redirection URL.

PHP Download php_to_rails/php/headers/redirection.php

```
header("Location: /documents/new");
```

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  redirect_to(:controller => "documents", :action => "new")
end
```

This redirect_to method can also be given a string if the redirection URL is outside the domain of the current application.

PHP Download php_to_rails/php/headers/redirection_external.php

```
header("Location: http://maintainable.com");
```

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  redirect_to('http://maintainable.com');
end
```

13.8 Security

There are various security concerns when developing Rails applications. Many of these you'll be familiar with from encountering the same issues in PHP. Others are unique to the conventions used in Rails.

Escape Output

You should always escape variables for output. This eliminates bugs because of improperly escaped entities but more importantly alleviates security concerns such as cross-site scripting attacks. The equivalent of PHP's htmlentities function in Rails is the h helper method. We can use this method just like any other helper method, and a common usage pattern is to leave off the parentheses when outputting a single variable within the Ruby interpreter. In this case, the h method is placed at the beginning of the tags used to open the Ruby interpreter such as <%=h.

PHP Download php_to_rails/php/security/escape_output.php

```
<div>
  <a href="/documents/show/<?= $document->id ?>">
    <?= htmlentities($document->filename, ENT_QUOTES) ?>
  </a>
</div>

<div>
  <?= htmlentities($document->contentType, ENT_QUOTES) ?>
</div>
```

Ruby Download php_to_rails/ruby/demo_1/app/views/examples/escape_output.html.erb

```
<div>
  <%= link_to h(@document.filename), :controller => "documents",
                                     :action     => "show",
                                     :id         => @document.id %>
</div>

<div><%=h @document.content_type %></div>
```

Filter Input

To avoid SQL Injection attacks in PHP, we always use a function such as mysql_real_escape_string to escape quotes and other potentially dangerous characters within a SQL statement. Rails accomplishes the same thing using replacement variables.

Any SQL fragment in our find statements can be stated as an array instead of a string. The first element is a SQL string with question marks as value placeholders. The rest of the array elements are values to be substituted into the string.

PHP Download php_to_rails/php/security/replacement_variables.php

```
mysql_connect('localhost', 'root', '');

$id   = isset($_POST['id'])   ? $_POST['id']   : null;
$name = isset($_POST['name']) ? $_POST['name'] : null;
$type = isset($_POST['type']) ? $_POST['type'] : null;

$query = sprintf("SELECT * FROM documents WHERE id='%s' LIMIT 1",
                 mysql_real_escape_string($id));

$query = sprintf("SELECT *
                    FROM documents
                   WHERE filename LIKE '%s'
                     AND content_type = '%s'",
                 mysql_real_escape_string("%$name%"),
                 mysql_real_escape_string($type));
```

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
  def my_action
    id, name, type = params[:id], params[:name], params[:type]

    # condition fragment
    doc = Document.find(:first,
                        :conditions => ["id = ?", id])
```

```
    # sql query
    docs = Document.find_by_sql(["SELECT *
                                    FROM documents
                                   WHERE filename LIKE ?
                                     AND content_type = ?",
                                 "%#{name}%", type])
  end

  def my_action
    begin
      @document = Document.find(params[:id])
    rescue ActiveRecord::RecordNotFound
      flash[:notice] = "Invalid document"
      redirect_to :action => :index
    end
  end

  def my_action
    @document = Document.find_by_id(params[:id])
  end

  def my_action
    # deliver the message
    NotificationMailer.deliver_confirm(@user)
  end

  def my_action
    # create, and deliver later
    email = NotificationMailer.create_confirm(@user)
    NotificationMailer.deliver(email)
  end

  def my_action
    render
  end

  protected
    def my_protected
      # this cannot be executed as an action
    end

end
```

Protect Attributes from Bulk Assignment

A common pattern used in Rails during form submission is to group together data for a particular object so that we can perform a bulk assignment in our controller. For example, we might have a Comment model such as the code on the next page.

Ruby Download php_to_rails/ruby/demo_1/db/migrate/004_create_comments.rb

```ruby
class CreateComments < ActiveRecord::Migration
  def self.up
    create_table :comments do |t|
      t.string  :email
      t.text    :content
      t.boolean :verified
      t.timestamps
    end
  end

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

Then the interface for a public form to create a comment might include only the email and content attributes, while displaying only the verified attribute for site administrators.

Ruby Download php_to_rails/ruby/demo_1/app/views/comments/new.html.erb

```ruby
<form method="post" action="/comments/create">
  <input type="text" name="comment[email]" />
  <input type="text" name="comment[content]" />
  <% if @user.admin? %>
  <input type="text" name="comment[verified]" value="1" />
  <% end %>
</form>
```

When the data for this is submitted, it will combine the data into a single hash that we can assign in the controller when creating our object.

Ruby Download php_to_rails/ruby/demo_1/app/controllers/comments_controller.rb

```ruby
def create
  @comment = Comment.new(params[:comment])
  if @comment.save
    flash[:notice] = 'Created successfully.'
    redirect_to :action => "index"
  else
    render :action => "new"
  end
end
```

The problem is that the verified attribute isn't actually secured for the model and is merely hidden from the view. There is nothing stopping a user from submitting this attribute through some other means such as curl, in which it would mark the comment as verified regardless of whether the user is an administrator.

Download php_to_rails/ruby/attr_protected.sh

```
curl -d "comment[verified]=1" http://localhost:3000/comments/create
```

The solution for this is to mark this attribute as protected from bulk assignment using the attr_protected method in our model.

Ruby Download php_to_rails/ruby/demo_1/app/models/comment.rb

```
class Comment < ActiveRecord::Base
  attr_protected :verified
end
```

We can also use a white-list approach, instead using attr_accessible to define the only attributes that are allowed during bulk assignment.

Ruby Download php_to_rails/ruby/demo_1/app/models/comment.rb

```
class Comment < ActiveRecord::Base
  attr_accessible :email, :content
end
```

Once we've secured our models this way, we need to remember that we now need to explicitly assign these attributes in our controller when they are applicable.

Ruby Download php_to_rails/ruby/demo_1/app/controllers/comments_controller.rb

```
@comment = Comment.new(params[:comment])
@comment.verified = params[:comment][:verified] if @user.admin?
```

Handle Missing Records

When we use the find method to load a record by primary key, it expects that the ID given is valid. When the ID given cannot be found, an ActiveRecord::RecordNotFound exception is raised. It is important to not trust that IDs given in the application are valid. Many times it is as easy as changing a number in a URL to throw an invalid ID into your action.

There are two ways of handling missing IDs. The first is to put your find within a begin/rescue block. How you deal with an invalid ID depends on the situation. Most of the time it is sufficient to simply redirect back to the index view with a polite message.

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  begin
    @document = Document.find(params[:id])
  rescue ActiveRecord::RecordNotFound
    flash[:notice] = "Invalid document"
    redirect_to :action => :index
  end
end
```

If you'd rather the object simply be Nil when the record is not found, you can use find_by_id instead of find. This usage is appropriate when you expect that the ID could not exist, and the code can continue to execute properly when the record is Nil.

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  @document = Document.find_by_id(params[:id])
end
```

Nonaction Controller Methods

All methods in a controller are assumed to be public actions unless stated otherwise. This means methods that were not intended to be accessed can be typed in the URL and cause errors in your application. The simplest way to prevent this is to give a protected visibility to any methods not intended to be actions within the controller.

Ruby Download php_to_rails/ruby/demo_1/app/controllers/examples_controller.rb

```
def my_action
  render
end

protected
  def my_protected
    # this cannot be executed as an action
  end
```

13.9 Debugging

The most popular debugging strategy in PHP is done using strategically placed print statements. Although there are certainly more sophisticated debugging solution for PHP, simply printing variables to the screen is usually pretty quick and efficient.

If you've tried to place print statements within your Rails controllers or models, you've probably noticed that they don't have any effect on the output sent to the browser. This is because any output generated in your Rails code has nothing to do with the data that Rails eventually renders to the browser. We do, however, have a few alternate strategies for debugging in Rails.

Logging Data

We'll usually use the logger in Rails to do simple debugging. Log files are written to the log/ directory in our application and are named based

on the current environment we are using. We discuss environments in more detail in Section 6.2, *Using Rails Environments*, on page 166. When you are working in the development environment, a lot of useful information is sent to the log automatically. This includes all the SQL executed and the list of parameters sent with each request. Simply viewing the log might give you enough information without further debugging.

Often you'll need to send further data to the log to inspect the contents of a variable. We can send data to the log using the logger.info method. This will work in models, controllers, and views. When you are logging objects, you'll probably want to use their inspect method to get a more useful output of their contents.

Ruby

Download php_to_rails/ruby/demo_1/app/controllers/meetings_controller.rb

```
def create
  @meeting = Meeting.new(params[:meeting])
  logger.info(@meeting.inspect)
  # ...
end
```

When we run this code, something similar to the following will be sent to our log.

```
#<Meeting id: nil, meets_on: "2007-11-30", location: "The Library",
  description: "Using OpenID", created_at: nil, updated_at: nil>
```

Interactive Debugging

Rails also has a sophisticated debugger based on the ruby-debug gem. To use this debugger, first install ruby-debug using gem install:

```
my_app> gem install ruby-debug
Building native extensions.  This could take a while...
Successfully installed ruby-debug-base-0.9.3
Successfully installed ruby-debug-0.9.3
2 gems installed
...
```

Once we've installed this required gem, we need to restart the server for our application using the --debugger option:

```
my_app> ruby script/server --debugger
=> Booting Mongrel (use 'script/server webrick' to force WEBrick)
=> Rails application starting on http://0.0.0.0:3000
=> Debugger enabled
...
```

Now in our application, we can initialize the debugger by placing debugger somewhere in our application.

Ruby

Download php_to_rails/ruby/demo_1/app/models/meeting.rb

```
# formatted name based on date
def name
  debugger
  meets_on.to_s(:long)
end
```

When the application reaches this point, it will invoke the interactive debugger.

```
/user_group/app/models/meeting.rb:23 meets_on.to_s(:long)
(rdb:5)
```

From here we can walk through the call stack and inspect our environment using various commands. To see a list of available commands, type help.

```
(rdb:5) help
ruby-debug help v0.9.3
Type 'help "command-name"' for help on a specific command

Available commands:
backtrace break catch cont delete display down eval exit finish frame
help irb list method next p pp quit reload restart save script set
step thread tmate trace undisplay up var where
```

Follow the guidelines here for using the help command to get additional information on the various commands. More detailed usage instructions can also be found on the Ruby-Debug website.[2]

13.10 Accessing the Database

We are quite familiar with writing SQL in PHP. While you are learning Rails, you may wonder how to query the database directly without using ActiveRecord objects. The short answer is that it's possible but not a good idea. ActiveRecord uses callbacks hooks and validations to ensure that the data entering the database adheres to the rules assigned in our model classes. Accessing and querying the database directly will circumvent all the logic we've added to the model layer of our application.

2. http://www.datanoise.com/ruby-debug/

With this in mind, there are sometimes performance reasons to bypass validations and callbacks. To perform mass updates, we can use the update_all method. The first argument is a SQL fragment with the updates to apply, and the second argument is the conditions.

```
def update_admin_for_nyc
  self.update_all("admin = 1", "location = 'NYC'")
end
```

We can perform a similar operation for mass deletions using the delete_all method. This method takes a single argument with the conditions on which to delete records.

```
def delete_from_tulsa
  self.delete_all("location = 'Tulsa'")
end
```

Most SELECT-based query operations can (and should) be done using the versatile find method. This method supports options such as :select, :from, :group, :limit, :offset, and :conditions.

```
def find_archives
  self.find(:all, :select     => "id, name",
                  :from       => "user_archives",
                  :conditions => "admin = 1",
                  :limit      => 10,
                  :offset     => 10)
end
```

If the find method is not capable of performing the query you need, you can drop down to using the find_by_sql method to query. This method works just like find(:all) but uses a complete SQL string.

```
def find_including_archives
  sql = "SELECT * FROM users UNION SELECT * FROM user_archives"
  self.find_by_sql(sql)
end
```

If you absolutely need to drop down to execute straight SQL, you can do this within your models using the connection.execute method.

```
def swap_to_archive
  connection.execute("INSERT INTO user_archives SELECT * from users")
end
```

Remember that using execute is usually a last resort. Do some research first to find whether there is a better way to accomplish what you are trying to do.

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 you stay on top of your game.

Visit Us Online

Rails for PHP Developers Home Page
http://pragprog.com/titles/ndphpr
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.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
http://pragprog.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: pragprog.com/titles/ndphpr.

Contact Us

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