




Addison-Wesley Professional Ruby Series

Rails Plugins

Extending Rails Beyond the Core

James Adam

What This Short Cut Covers	4
Section 1: Introduction.....	5
Section 2: Installing Plugins.....	14
Section 3: Saluton, Mondo! (init.rb)	33
Section 4: Sharing Code (lib).....	39
Section 5: The Rest (README, install.rb, Rakefile, tasks, and More)	51
Section 6: Plugin Development.....	62
Section 7: Sharing Classes and Code.....	83
Section 8: Testing Plugins	100
Section 9: Sharing Your Plugin	113
Section 10: Conclusions	121
About the Author.....	123



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this work, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this work, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Visit us on the Web: www.awprofessional.com

Copyright © 2007 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
75 Arlington St., Suite 300
Boston, MA 02116
Fax: (617) 848-7047

This product is offered as an Adobe Reader™ PDF file and does not include digital rights management (DRM) software. While you can copy this material to your computer, you are not allowed to share this file with others.

ISBN 0-321-48351-0

First release, September 2006

TABLE 3.1 Special Methods Available in init.rb

Method	Description	Example
name	The name of the current plugin (i.e., the name of the directory in /vendor/plugins).	"hello_world"
directory	The directory in which this plugin exists.	"script/../config/../vendor/\nplugins/hello_world/lib"
loaded_plugins	A Set containing the names of all plugins already loaded, including this one.	#<Set: {"hello_world"}>
config	The Configuration object created in environment.rb.	See the Rails::Configuration class in the Rails source code for more information

While this plugin certainly isn't the most revolutionary piece of programming, it's sufficient to demonstrate that every time Rails loads, the `init.rb` files for each plugin are loaded. We can actually be more specific than this: **every time the Rails environment is loaded, your plugin's `init.rb` files will be evaluated, if they exist.** This includes when your application starts in a server, or even when you are running code directly via `script/runner`:

```
$ script/runner "puts 'Rails is ready.'"
```

Current Rails version: 1.1.4
 Rails is ready.

In general, if you need your plugin to do *something* every time Rails loads, then the code that performs this action is placed in the `init.rb` file. We're going to use this file frequently, as you'll see later.

Summary

This section introduced plugin creation and highlighted the importance of the `init.rb` file. If your plugin needs to perform some setup behavior each time the application server starts, that functionality will ultimately be initiated from code in this file.

SECTION 4

Sharing Code (lib)

It's great being able to display information when your application starts, but typically you'll want to enhance Rails applications in more interesting and useful ways than that. Most commonly, you'll want to make one or more methods or classes available to the rest of your application.

Section 4: Sharing Code (lib)

The lib Directory	40
The CopyrightHelper Plugin.....	42
Rails, Modules, and Auto-Loading Code.....	44
Using the Copyright Plugin in an Application	45
Including Code Automatically	48
Adding to ApplicationHelper	48
Summary	50

The lib Directory

The two defining aspects of a plugin are the presence of the `init.rb` file and of a directory in the plugin called `lib`. If neither of these exists, then Rails will not recognize the subdirectory of `vendor/plugins` as a plugin. This section discusses the `lib` directory.

In the previous section, we saw that the `init.rb` file is evaluated each time the Rails environment is loaded. However, before Rails even evaluates `init.rb`, it adds the `lib` directory to Ruby's load path. You can verify this by opening a console session and searching the load path for an entry that matches the plugin's `lib` directory:

```
>> $LOAD_PATH.select { |p| p =~ /hello_world\/lib$/ }  
=> ["script/../../config/../../config/../../vendor/plugins/hello_word/lib"]
```

Adding this directory to the load path means that Ruby can now load source files from this directory using the `require` statement.

A Simple Example

Typical application development often included creating little classes that encapsulate some small, useful behavior. For example, it might often be useful to be able to create **deep copies** of objects. A deep copy is the duplication of an object and all its attributes, so that no change in the copied object will change any attribute in the original.

The following is an example of why `Object#dup` (which performs **shallow copying**) can be problematic:

```
>> a = [1]; b = [a]  
=> [[1]]
```

SECTION 4

The lib Directory

```
>> c = b.dup # perform a shallow copy
=> [[1]]
>> b[0][0] = :changed # now change something in b
=> [[:changed]]
>> c
=> [[:changed]] # c has also been changed!
```

While this example is fairly trivial, it's not hard to imagine situations where shallow copying could cause unexpected results. To avoid this, you can add a new method to the `Object` class that lets you perform deep copying (see Listing 4.1).

LISTING 4.1 Adding `deep_copy` to the `Object` class

```
vendor/plugins/misc_stuff/lib/deep_copy.rb

class Object
  def deep_copy
    Marshal::load(Marshal.dump(self))
  end
end
```

Since you just created a file called `deep_copy.rb` in the `lib` directory, you can now use it in your application by require-ing it:

```
>> require 'deep_copy'
=> true
>> [1].deep_copy # to prove deep_copy is available
=> [1]
```

Adding `require 'deep_copy'` to the plugin's `init.rb` file will ensure that the `deep_copy` method is available to every object in your Rails application.

The Code Payload

You can bundle *any* class or Ruby code in a plugin's `lib` folder and then load it (or allow other developers to load it) using Ruby's `require` statement. This is the simplest way to share Ruby code among multiple Rails applications.

The CopyrightHelper Plugin

Imagine you're developing several applications that require a similar copyright line at the bottom of every page:

© 2006, 43indicators LLC. Click [here](#) for Terms of Service.

Rather than type this into each application, you can save some effort and write a Ruby method to produce this string automatically. Listing 4.2 shows the beginnings of a simple implementation of this.

LISTING 4.2 A copyright message method

```
def copyright_notice(company_name)
  "&copy; #{Date.today.year}, #{company_name}."
end
```

How do we make this method available in the views in our Rails application? First, create a new plugin:

```
$ script/generate plugin CopyrightHelper
  create vendor/plugins/copyright_helper/lib
  <...>
```