Practical Ruby Gems

Copyright © 2007 by David Berube

ISBN-13 (pbk): 978-1-59059-811-5

ISBN-10 (pbk): 1-59059-811-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section.

PART 1

Using RubyGems

This section of the book introduces RubyGems and explains how you can start using them in your code.

CHAPTER 1

What Is RubyGems?

In short, RubyGems lets you distribute and install Ruby code wherever you can install Ruby.

Specifically, RubyGems is a package-management system for Ruby applications and libraries. It lets you install Ruby code—called *gems*—to any computer running Ruby. It can resolve dependences for you, so if you want to install a given piece of software, RubyGems can handle that for you. It can even resolve version dependencies—so that if a certain gem or your code requires a certain version of another gem, it can take care of that. It also wraps all of this functionality in a very easy-to-use package.

The gems come in a variety of types. For example, if you had a Web application to which your users uploaded pictures from a digital camera, you'd likely need to resize the pictures, which come in a variety of sizes. You could write the resizing code by hand, but it'd be considerably faster to use the `rmagick` gem to resize the pictures—and you could add additional features like cropping, rotation, sharpening, and so on with just a few extra lines of code, since `rmagick` includes all of those features. (See Chapter 25 for more details.)

Alternatively, if you want to develop a Web application using `Ruby on Rails`—which is a full-featured, very powerful Model View Controller (MVC) Web framework—you could install that using RubyGems as well. `Rails` consists of a number of libraries and utilities—all of which can be installed by RubyGems with just one command. (See Chapter 23 for more information.)

This chapter covers the features of RubyGems and how it differs from other package-management systems.

Why Use RubyGems?

First of all, RubyGems makes it easy use to install Ruby software. For example, Instiki (`http://instiki.org/`) is a wiki—a kind of content-management system—and if we wanted to install the `instiki` gem, we could do so with the following command in the Linux/Mac OS X shell or the Windows command prompt:

```
gem install instiki
```

Of course, to do that you'd need RubyGems installed, and we'll cover that in the next two chapters. For now, though, you can see how easy it is to install gems—just one command and RubyGems takes care of the rest.

This can be extremely important; for example, if you had a Web application written in Ruby and your server failed, you'd need to be able to quickly and easily install all of the software that your application needs on a new server.

It Provides a Standard Way to Describe Ruby Software and Requirements.

RubyGems lets you define *gemspecs*. A gemspec describes software—it includes the name, version, description, and so forth. This gemspec can be built into a `.gem` file, which is a compressed archive containing the gemspec and all of the files that the software requires.

This `.gem` file can be uploaded to RubyForge, which lets you install it from any Internet-connected RubyGems installation, or it can be distributed via traditional means, like HTTP or FTP. Because the `.gem` file contains a description of the program, you can also use the `gem list` command to see the details of the gem or to search for similar gems. (You can find more details on the `gem list` command in Chapter 3 and you can find out more about building gems in Chapters 34 and 35.)

For example, if you upgrade a version of a gem and the new version has additional requirements, you won't need to scour the documentation for the changes—RubyGems will automatically read the requirements from the gemspec since the format of the gemspecs is standard.

It Provides a Central Repository of Software.

One of the aspects of RubyGems that makes it so appealing is it gives you access to RubyForge —a central repository of Ruby software. You can find out more about RubyForge at, `http://rubyforge.org`.Without RubyForge, you'd have to locate, download, and then install a gem and its dependencies. With RubyForge, though, RubyGems can automatically locate the software and its dependencies for you.

Although most Rubyists (Ruby programmers and enthusiasts) install gems only from the central repository, you aren't required you to use it—you can install gems from any location you choose. (You can also set up your own gem server, which you'll learn about next.) For example, if you had to move your software from one operating system to another, your operating system's packaging system and repository would be different, but RubyGems would stay the same—you can use RubyForge wherever RubyGems is installed.

It Lets You Redistribute Gems Using a Gem Server.

The technology used to serve gems comes with RubyGems. You can set up your own RubyGems server on a local network or on the Internet without much trouble; if, for example, you'd like to cache all of the gems your development team uses on a local server to speed up downloads, you can do that.

If you'd prefer not to use RubyForge and rather distribute gems via your own website or gem server, you can do that too. You can find more details in Chapter 35.

It Handles Software Dependencies for You.

RubyGems can take care of dependencies automatically. That means that when you install a gem, it can automatically determine what other gems are required and ask you if you'd like to install them.

This can make your life much easier, since a significant amount of Ruby software is built using other Ruby software—and that other Ruby software might require still more software. Without RubyGems, you might have to spend hours installing and researching dozens of packages to get complex software working. With RubyGems, you can just install the gem and let it resolve dependencies for you.

It Handles Multiple Software Versions Intelligently.

RubyGems can store multiple gem versions, and software that uses RubyGems can request particular gem software versions—so, for example, an application that requests the `ActiveRecord` gem (`http://rubyforge.org/projects/activerecord/`) could request a gem that's newer or older than a given version. This is very helpful if, for example, a later version of a gem breaks your program, or if your program requires a feature from the latest version of a gem. (You can find more details on how to do this in Chapter 4.)

It Can Be Used Transparently in Place of Regular Ruby Libraries.

RubyGems has a facility that makes it transparent to use gem software. For example, suppose you wanted to use the Camping (`http://rubyforge.org/projects/camping/`) Web microframework. If you installed Camping the traditional way, you would use the Camping library in your code like this:

```
require 'camping'
```

If you installed it via RubyGems, you use the exact same code. As you can see, using code via RubyGems is transparent, so you can switch back and forth easily; if you distribute your software, the user does not need to have RubyGems installed—only the required library.

Note, however, that if you want to require that a certain version of the software is installed, you'll need to use a special RubyGems statement in your code. (See Chapter 4 for further details.)

It Lets You Use the Same Technology on Any Operating System.

RubyGems targets all platforms that run Ruby. If it runs Ruby, it runs RubyGems. A number of other systems exist to make software installation easier; there's everything from those that simply install software—like Window's MSI installation system—to full package-management systems, like Debian Linux's apt (Advanced Package Tool), Red Hat's yum, or OS X's DarwinPorts. Such systems are generally operating system–dependent, though, as we'll discuss next.

How Does RubyGems Compare to Other Packaging Systems?

Operating system–specific packaging systems, such as apt or yum, can carry Ruby software as well. Since Ruby software can be used by non-Rubyists, this is important. It's also convenient if you need just a few pieces of software. For Rubyists, though, it's usually better to install gems using RubyGems, since RubyGems has the best selection of Ruby software and the latest versions. Additionally, unlike RubyGems, OS-native packagers don't handle multiple gem versions installed simultaneously.

However, it is possible to install a limited selection of Ruby software using other packaging systems. For example, you could install the MySQL Ruby bindings via gem like this:

```
gem install mysql
```

Alternatively, you could install the same library via apt-get under Ubuntu Linux like this:

```
apt-get install libmysql-ruby1.8
```

Finally, you could install it via DarwinPorts under OS X like this:

```
port install rb-mysql
```

Note that those three commands require you to be logged in as `root`—if you prefer, you can prefix each command with `sudo`, which will execute that single command with the root-user privileges.

In some cases you can install software via apt or another packaging system. Such systems usually have a very limited selection of Ruby packages, but if they happen to include all of the software you need, you may be able to use them. Consult the documentation that comes with your Linux distribution or other packaging system.

Note, though, that installing gems from your OS distribution is not recommended. It means you have to use the version of the software in your OS's repository, and often this lags significantly behind the RubyGems versions. Using the gem installer, as we do throughout this book, will automatically give you access to the most recent gem versions.

Of course, you can always skip package-management systems entirely—you can install Ruby software by running an install script manually or by copying files into the `lib` directory of your Ruby installation. If you want to do so, download the software you want from its homepage and consult the included `README` or `INSTALL` file.