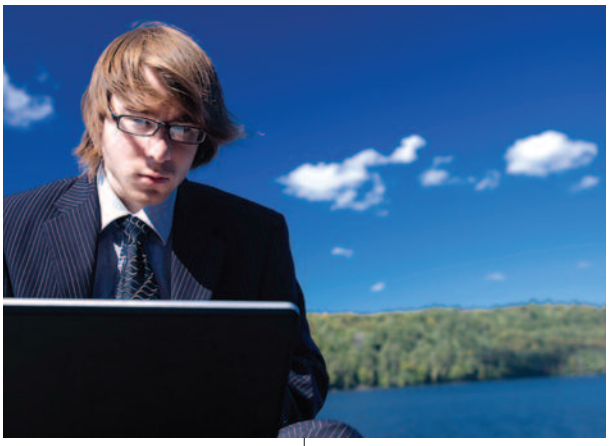
A Java Developer's Guide to Ruby

By Mark Watson

As a Java developer, why should you learn Ruby? Because Ruby's versatility and flexibility complement Java well, and you will be a more effective and efficient developer if you use both languages. In fact, I use Java, Ruby, and Common Lisp for all my development and Ruby has become a core part of my work life. Specifically, the following reasons make Ruby compelling for Java developers:



Jupiterimages

- As a scripting language, Ruby is an effective tool for small projects. When I need to write utilities for data conversion and text processing quickly, I almost always use Ruby.
- Ruby is a dynamic and terse language.
- Using Ruby will often offer a different perspective on problem solving.
- JRuby is still a work-in-progress but I believe it eventually will provide an excellent Ruby deployment platform using the Java VM. Currently, IntelliJ, NetBeans, and Eclipse all provide excellent Ruby support.
- As the cost of software maintenance is roughly proportional to the number of lines of code, and Ruby programs are short and concise, they tend to be easier to read, understand, and maintain.
- The Ruby on Rails Web development framework is great for small and medium-sized database-backed web applications. You need to know Ruby if you want to use Ruby on Rails.

To demonstrate why Ruby is a good fit for Java developers, this article introduces the language features that will make you more efficient (see Table 1. Ruby and Java Feature Comparison) and then shows short program examples in both languages.

> Because Ruby's versatility and flexibility complement Java well, and you will be a more effective and efficient developer if you use both languages.

Table 1. Ruby and Java Feature Comparison

Language Features	Ruby	Java
Extending All Classes	Yes	Non Final Classes Only
Duck Typing	Yes	No
Code Blocks	Yes	No
Regular Expressions	Native	Standard Library Support
Supports Using External Programs	Yes	Yes, but not as easily as Ruby
Network Programming	Standard Library Support	Standard Library Support
Typing Dynamic	Static	
Class Inheritance	Support mix-ins from multiple classes	Single
String Handling	Yes	Yes

What You Need

To follow along with the rest of the article, you need to install external Ruby libraries. The RubyGems library system makes this easy. Download it from RubyForge and follow the installation instructions for your operating system. (If you already have Ruby set up, you can verify that your setup includes RubyGems—many Ruby install packages do—by typing gem in a command shell to check for installation.) Having a central repository for libraries and a standard tool like RubyGems will save you a lot of time: no searching for the libraries you need, installing them, and using them in multiple projects.

Use the following commands to install the required gems:

```
gem query --remote # if you want to see all available remotely installable gems
sudo gem install activerecord
sudo gem install mysql # if you want to use MySQL
sudo gem install postgres-pr # optional: install "pure ruby" PostgreSQL interface
sudo gem install postgres # optional: install native PostgreSQL interface
sudo gem install ferret # a search library like Lucene (same API)
sudo gem install stemmer # a word stemming library for demonstrating extending a
class
gem query # to show gems locally installed
gem specification activerecord # info on gem (ActiveRecord in this example)
```

Under Mac OS X and Linux, you will need to run the gem installs using sudo; if you are a Windows user, remove "sudo" from the previous commands.

This article also assumes that you will open a Ruby irb shell as follows and keep it open while you're reading:

```
markw$ irb
>> s = "a b c"
=> "a b c"
>>
```

The example programs and code snippets are short enough to copy and paste into an irb interactive session.