Shay Friedman

IronRuby

UNLEASHED

SAMS

IronRuby Unleashed

ISBN-13: 978-0-672-33078-0
ISBN-10: 0-672-33078-4

Library of Congress Cataloging-in-Publication Data

Friedman, Shay.
IronRuby unleashed / Shay Friedman.
p. cm.
ISBN 978-0-672-33078-0
1. IronRuby (Computer program language) 2. Microsoft .NET Framework. 3. Ruby (Computer program language) I. Title.
QA76.73.I586F74 2010
006.7'882—dc22

2009050114

Printed in the United States of America

First Printing: February 2010

Trademarks

Warning and Disclaimer

Bulk Sales

Editor-in-Chief
Karen Gettman

Executive Editor
Neil Rowe

Acquisitions Editor
Brook Farling

Development Editor
Mark Renfrow

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Keith Cline

Indexer
Word Wise Publishing Services

Proofreader
San Dee Phillips

Technical Editor
Justin Etheredge

Publishing Coordinator
Cindy Teeters

Interior Designer
Gary Adair

Cover Designer
Gary Adair

Compositor
Nonie Ratcliff

Introduction

The Ruby language was developed by Yukihiro Matsumoto, who began developing Ruby on February 24, 1993. His main reason for doing so was his dissatisfaction with the scripting languages at the time, languages such as Perl and Python. He designed Ruby to be intuitive, to be natural, and to follow the "principle of least surprise"—making developers enjoy writing code and focus on the creative part of programming instead of fighting the language to fit their needs.

Ruby 1.0 was released on December 25, 1996, exactly 1 year after the first public release (version 0.9.5) of Ruby. For the first year afterward, Ruby was mainly used inside Japan. Its use expanded outside of Japan a few years later, but it was still used by a small number of eager early adapters.

In 2006, David Heinemeier Hansson from 37signals released a web development framework named Ruby on Rails. This innovative MVC web framework made the difference. More and more developers started using Ruby on Rails to develop their web applications and in the process became familiar with the Ruby language, too. Following these newcomers, a phrase was coined to explain how most current Ruby developers have come to use it: I came for the Rails, but I stayed for the Ruby.

Since then, Ruby has become one of the most popular programming languages in the world and is being used by thousands of developers every day.

Ruby is a dynamic language. It combines ideas from Perl, Smalltalk, Eiffel, Ada, and Lisp to provide an intuitive, flexible, and simple-to-use language. Its strengths are in its permissive syntax and powerful built-in capabilities, especially metaprogramming capabilities. However, it never appealed to .NET developers because it lacked integration with .NET code. This situation has changed with the birth of IronRuby.

IronRuby is Microsoft's implementation of the Ruby language. It runs on top of the Dynamic Language Runtime (DLR), which is a special dynamic language service provider that is built on top of the Common Language Runtime (CLR).

IronRuby provides seamless integration with .NET code. It enables you to use .NET objects in Ruby code just as if they were pure Ruby objects. This opens vast opportunities to the .NET world and to the Ruby world. Both sides gain the power and strength of the other and provide together a new and exciting development environment.

In this book, I take you through all aspects of IronRuby so that you learn how to best leverage the language for the simplest of tasks to the most advanced ones.

Part I, "Introduction to IronRuby," is an overview of the different pieces that make IronRuby possible. You learn about where it all began and the main concepts of the Ruby language, the .NET Framework, and the DLR. Part I ends with a chapter that introduces you to IronRuby for the first time. In that chapter, you learn the basics about using IronRuby and witness the powerful capabilities it brings to your development environment.

Part II, "The Ruby Language," is devoted to an in-depth tutorial of the Ruby language. The part starts with the basic syntax, goes on with Ruby object-oriented programming capabilities, and ends with advanced concepts and techniques.

In Part III, "IronRuby Fundamentals," I add the Iron to Ruby. This part contains all the information you need about IronRuby .NET integration. I explain how every .NET item can be used from IronRuby—from variables to implementing .NET interfaces.

Part IV, "IronRuby and the .Net World," is the practical part. It contains guides for how to use IronRuby in several different scenarios. Most of the current .NET and Ruby frameworks are explained, including WPF, ASP.Net MVC, Ruby on Rails, and Silverlight. In addition, other possible usages are explained, such as testing .NET code using Ruby's different unit testing frameworks and running IronRuby code from .NET code.

The last part of the book, Part V, "Advanced IronRuby," covers IronRuby advanced topics. If you want to extend IronRuby objects or to create .NET code libraries that fit better to Ruby code, you will be interested in what this part has to offer.

I believe that IronRuby can enhance your work and enable you to do things you have not done before. I hope you find this book helpful and informative and that you can exploit its contents in your own projects and development tasks.

CHAPTER 4

Getting Started with IronRuby

IN THIS CHAPTER

- Overview
- Installing IronRuby
- Executables and Tools
- Development Environments
- The Power of IronRuby

IronRuby is Microsoft's implementation of the Ruby language on top of the DLR. Its main goal is to provide seamless interoperability between Ruby and the .NET Framework.

IronRuby combines the powers of both the .NET Framework and the Ruby language. On the one hand, it contains the built-in capabilities of Ruby, and on the other hand, it is capable of using the wide variety of frameworks and libraries of the .NET Framework. The combination opens a whole new set of opportunities to both Ruby and .NET developers.

In this chapter, you get your first taste of IronRuby. You install it, read an overview of the language and its tools, and start to discover the power it brings to the .NET family.

Overview

IronRuby is Microsoft's implementation of the Ruby programming language. It is built on top of the DLR and provides seamless integration between Ruby code and .NET Framework code. It is compatible with Ruby 1.8.6 and runs on .NET Framework 2.0 Service Pack 1 and above.

IronRuby was first announced on April 30, 2007, at the MIX conference. *Iron*, in its name, is actually an acronym and stands for "implementation running on .NET."

IronRuby is supported by the Common Language Runtime (CLR) and Mono, which means that it can be run on Windows, Linux, UNIX, BSD, Mac, and all other operating systems that are supported by Mono. Apart from operating

systems, IronRuby can also be run from the browser using Silverlight. (See Chapter 16, "Silverlight," for more about IronRuby and Silverlight.)

IronRuby is an open source project and is released with full source code under the Microsoft Public License (MS-PL). The code is hosted on GitHub and can be downloaded from the CodePlex site, too. Because it is an open source project, the IronRuby team is looking for contributions both in bug fixing and library implementation. Look at the contribution page on IronRuby's GitHub home page to see how you can help (http://wiki.github.com/ironruby/ironruby/contributing).

Installing IronRuby

IronRuby runs on .NET Framework 2.0 Service Pack 1 and above. Therefore, you need to have .NET Framework 2.0 SP1 or above installed on your machine before you start. Same goes for every machine on which you deploy your IronRuby applications.

You can download .NET Framework 2.0 SP1 from http://www.microsoft.com/downloads/details.aspx?familyid=79BC3B77-E02C-4AD3-AACF-A7633F706BA5.

When you have the correct framework, we can move on and install IronRuby. The recommended method of installing IronRuby is by using the IronRuby installer. Follow the next steps to do so:

1. Visit http://www.ironruby.net/download and click the Download IronRuby link.
2. The downloaded file is an MSI file. Double-click it to start the installation.

   The first input you will be asked to enter is the installation folder for IronRuby. The default is your program files directory\IronRuby. The folder you choose will be referenced as the IronRuby installation folder throughout this book.

   During the installation, you will be asked to select the features to install. The features that will be presented are as follows:

   - **Runtime:** The main files of IronRuby. This feature is required.
   - **Standard Library:** Ruby standard libraries. Needed if the standard libraries are used in IronRuby code.
   - **Samples:** Sample IronRuby applications like WPF and PowerShell samples.
   - **Silverlight Binaries:** Binaries needed for using IronRuby in Silverlight applications.
   - **Add IronRuby to %PATH%:** Adds IronRuby binaries path to the PATH environment variable. This spares the need to provide full path to IronRuby executables when they are called from the command line.
3. After approving all steps, the installation of IronRuby takes place.

Another option to install IronRuby is manually. IronRuby can be downloaded as a zip package. This lets you fully control the installation process but also forces you to execute the automatic tasks manually (optionally).

Follow the next steps to install IronRuby manually:

1. Visit IronRuby's CodePlex homepage at http://ironruby.codeplex.com and click on the Downloads button on the page menu. In the downloads page, choose to download the IronRuby ZIP package.
2. After the download is complete, extract this Zip file to the folder in which you want to place IronRuby (for example, C:\IronRuby). This folder will be referenced as the IronRuby installation folder throughout this book.

You're actually done now and can start using IronRuby. However, if you want IronRuby to be available from every location on your Windows system, you want to add the <installation folder>\bin path to the Windows PATH environment variable. Be aware, however, that this can be done only if you have administrative privileges.

To do that, follow the next steps:

1. Navigate to Start > My Computer and right-click My Computer. On the menu, choose Properties as presented in Figure 4.1.

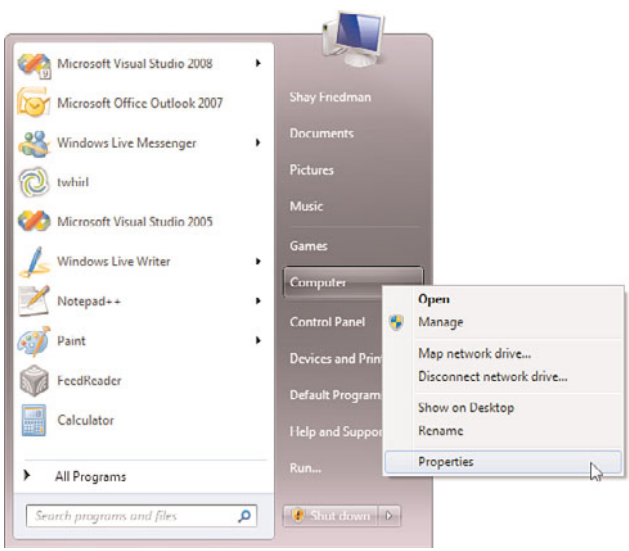

FIGURE 4.1 My Computer Properties Menu Item.

2. In the open dialog, click Advanced System Settings as presented in Figure 4.2.

FIGURE 4.2 The Advanced System Settings Link.

3. Click the Environment Variables button as presented in Figure 4.3.

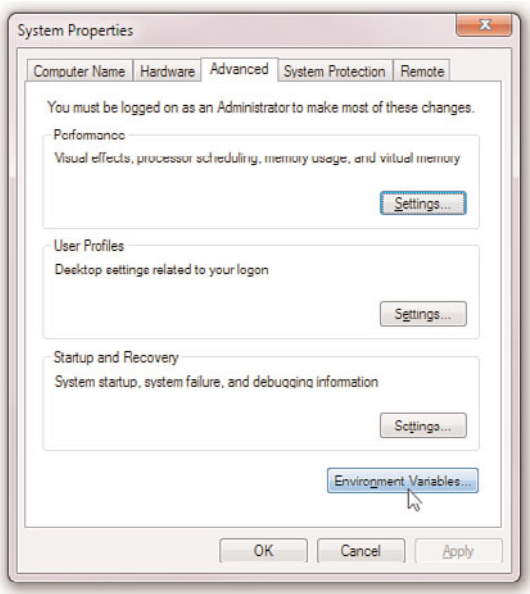

FIGURE 4.3 The Environment Variables button.

4. Find Path in the System Variables section (the lower part), select it, and click Edit as presented in Figure 4.4.
5. In the Edit System Variable dialog, place the cursor at the end of the Variable Value field and add a semicolon (;) and **<IronRuby installation folder>\Bin**. For example, if you've extracted IronRuby to C:\IronRuby, you add **;C:\IronRuby\Bin** to the Variable Value field as presented in Figure 4.5.

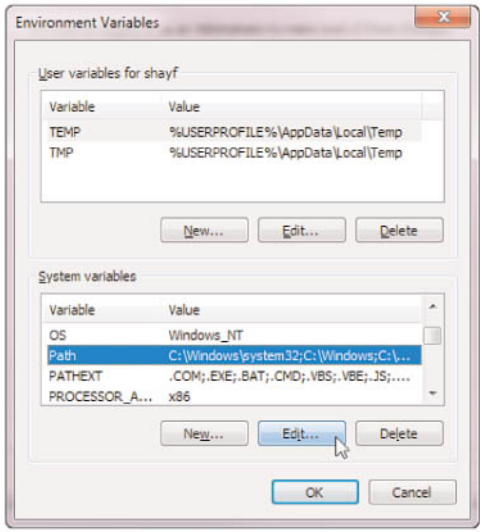

FIGURE 4.4 The PATH Environment Variable.

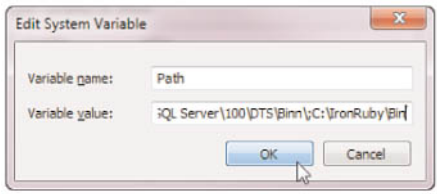

FIGURE 4.5 Setting the PATH Environment Variable.

6. Click OK on all the dialogs you've opened during the process to save the new setting.

Congratulations, IronRuby is now installed on your machine.

IronRuby Folders

After you extract IronRuby to the desired installation folder, you notice several folders there. Table 4.1 lists and describes the folders.

Getting the Sources

If you'd like to go deep into IronRuby and go through its code, you need to download the source code of IronRuby. The source code is hosted on GitHub and uses Git as its source control application. You can download the source code in a Zip format from http://github.com/ironruby/ironruby/zipball/master.

TABLE 4.1 IronRuby Folders and Their Roles

Folder	Description
Root	Contains license files, release notes, and the package readme file
Bin	Contains the IronRuby binaries, executables, and tools
Libs	Contains the standard libraries, including special IronRuby libraries and the RubyGems repository
Samples	Contains a few IronRuby samples and the IronRuby tutorial application
Silverlight	Contains binaries, samples, and tools for embedding IronRuby in Silverlight

To download the sources, you do not have to install Git on your machine. You need to do so only to contribute to the IronRuby code. For more information about how to contribute and how to use Git, look at the IronRuby wiki at http://wiki.github.com/ironruby/ironruby.

Executables and Tools

IronRuby comes with several different executables and tools. All of them are located under the Bin folder in the IronRuby installation directory.

Table 4.2 lists and describes the executables and tools in the IronRuby Bin folder, and then the following subsections take a closer look at them.

TABLE 4.2 IronRuby Executables and Tools

File	Description
ir.exe and ir64.exe	The main IronRuby executable. It is the IronRuby interpreter file. ir64.exe is for 64-bit systems.
iirb.bat	The IronRuby REPL (read-evaluate-print loop) console.
igem.bat	Used to work install and manage RubyGems. See Chapter 8, “Advanced Ruby.” for more information about RubyGems.
irackup.bat	Runs Rack, which is a Ruby framework that simplifies the way of interacting with different Ruby web servers.
irake.bat	Executes the Rake. See Chapter 8 for more about Rake.
irails.bat	Used to create a Ruby on Rails application. See Chapter 14, “Ruby On Rails.” for more about Ruby on Rails.
irdoc.bat	Runs RDoc, which is a Ruby tool to create formatted documentation out of Ruby code. The output can be plain text or a formatted HTML.
iir.bat	Ruby tool to read the textual documentation of Ruby objects. (The documentation is created by RDoc.)

The IronRuby Interpreter (ir.exe)

The IronRuby interpreter is the heart of IronRuby. Everything goes through it. For example, all the tools mentioned in Table 4.2 eventually run ir.exe.

The IronRuby interpreter can run Ruby files as well as a REPL console. These are two different modes, and so I discuss each of them separately.

REPL Console Mode

The REPL console mode opens a console where you can write code and execute it immediately.

To run IronRuby in the REPL console mode, follow these steps:

1. Click Start > Run.
2. Type **cmd** and click OK. The command prompt opens.
3. If you haven't added the IronRuby installation folder to the Path system variable, navigate to <installation folder>\Bin. If you have updated the Path system variable, skip this step.
4. Type **ir** and press Enter.

The IronRuby REPL console opens. You can now write Ruby code there (for example, "**puts 'hello world'**") or even write whole classes and use them.

The format is simple. Each line where you can write a Ruby statement starts with a triple right-angle sign (>>>). If the line is not assumed to be continued (like method or class definitions), when you press Enter its output (if any) is printed to the console and the next line contains the return value of the statement preceded by an equal or greater than sign (=>). If the line is assumed to be continued, the next statement line starts with an ellipsis (...).

THE RETURN VALUE OF METHODS THAT DO NOT RETURN ONE

You notice that after executing methods with no return value, such as puts, the console still shows a return value: `nil`. This is really the return value.

Every method in Ruby returns a value. If the method code doesn't return a value, the method returns `nil` to the caller.

Figure 4.6 shows an REPL console session.

The REPL console mode has some specific command-line switches that can be used when running ir.exe. Table 4.3 lists and describes the switches. Be aware that the switches are case sensitive.

These are the REPL mode-only switches. All other switches appear on Table 4.4, and some can be used in this mode, too.

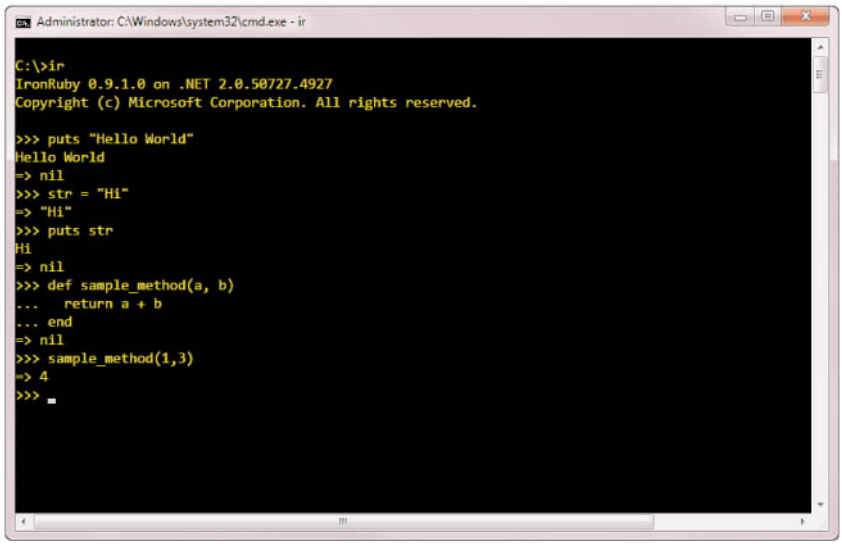

FIGURE 4.6 A screenshot of an IronRuby REPL console session.

TABLE 4.3 ir.exe REPL Mode Command-Line Switches

Switch	Description
-X:ColorfulConsole	Makes the REPL console colorful. Prompt signs such as >>> and ... appear gray, errors red, and warnings yellow; messages (like the banner on top) appear cyan, and all other output uses the default console output color.

File Execution Mode

This mode executes a given file. Its format is as follows:

```
ir [options] [file path] [file arguments]
```

All switches and ir.exe command-line arguments should be placed before the file path. Otherwise, they will be considered arguments of the file and will be passed to it rather than to the interpreter.

The simplest way to execute an IronRuby file is by passing only its path to the interpreter. For example, the next command executes the file test.rb, which is located within the current directory:

```
ir test.rb
```

Along with this way, IronRuby provides several command-line arguments that affect the way the code executes. Table 4.4 lists and describes available arguments for the ir.exe file execution mode.

TABLE 4.4 ir.exe Command-Line Arguments for File Execution Mode

Argument	Description
-d or -D	Debug mode. Allows using breakpoints in Ruby code with the Visual Studio debugger.
-e “command”	Executes the command and exits. Several -e arguments are allowed. When used, the file path should not be passed and will be ignored if it exists. For example: ir -e “str = ‘Hello World’” -e “puts str”
-I “directory”	Includes the given directory in the $LOAD_PATH variable. This means that it will be included in the search paths for required libraries.
-r “library”	Requires the library before executing the file. For example: ir -r “csv” test.rb
-v	Prints the IronRuby version on the first line.
-w	Turns on warnings in the default level (verbose).
-W[level]	Sets the warning level. 0 = silence, 1 = medium, 2 = verbose (default). For example: ir -W1 test.rb
-K[kcode]	Specifies KANJI (Japanese) code set. E or e = EUC, S or s = SJIS, U or u = UTF8. For example: ir -KU test.rb
-trace	Enables Ruby tracing capabilities.
-profile	Enables profiling. When this switch exists, a profile.log file will be added to the directory of the executed file with profiling information about the latest execution.
-18 or -19 Or -20	Run IronRuby in Ruby 1.8 compatibility mode (default), Ruby 1.9, or Ruby 2.0 compatibility mode accordingly. Ruby 2.0 doesn’t currently exist, so this switch is for future release only.

TABLE 4.4 ir.exe Command-Line Arguments for File Execution Mode

Argument	Description
-X:ExceptionDetail	In this mode, every exception is presented with a full call stack.
-X:NoAdaptiveCompilation	Disables adaptive compilation feature. This affects performance (for the worse).
-X:PassExceptions	In this mode, exceptions are not caught by the interpreter. This means that in case of an exception with no handling in the script, the application crashes.
-X:PrivateBinding	Enables binding to private members of CLR objects. Chapter 9, ".Net Interoperability Fundamentals," discusses the uses of this switch.
-X:ShowClrExceptions	When this mode is on, a CLR exception part is added to every exception with full exception details (the `exception.ToString` output).
-X:CompilationThreshold	Specifies the number of iterations before the interpreter starts compiling the code. Should be followed by a number. Note that this switch can affect performance dramatically. Hence it is not recommended to use it when not needed.
-h	Shows all available command-line arguments with a short description. When this exists, the file or REPL console does not run.

Development Environments

Support for IronRuby in Visual Studio is not available in IronRuby 1.0. Such support is not in Microsoft's current plans, and no one can really promise it will be in the near future.

However, the Ruby language already has several IDEs available. This section discusses some of them so that you can choose the one that best fits your needs.

Ruby in Steel

This commercial add-on to Visual Studio by SapphireSteel makes developing Ruby applications inside Visual Studio much more natural. It adds new Ruby project types, intellisense, code snippets, and syntax highlighting.

Figure 4.7 is a screenshot from Visual Studio that shows the syntax highlighting and intellisense capabilities of Ruby in Steel.

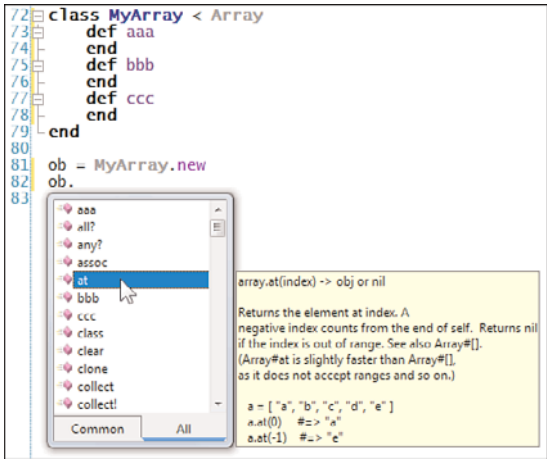

FIGURE 4.7 Ruby in Steel syntax highlighting and intellisense.

Although you cannot run IronRuby with Ruby in Steel out of the box, it is possible to alter the solution settings to execute ir.exe. Follow these steps to do so:

1. Inside Visual Studio, click Project > Project Settings.
2. In the Settings window, you see a Ruby region with a "Ruby Interpreter" line. Change the value on this line to the path of ir.exe.

   For example, if you installed IronRuby in C:\IronRuby, you must set the value to **"C:\IronRuby\Bin\ir.exe"**.
3. Save the project and press Ctrl + F5 to execute the Ruby files.

Ruby in Steel is a commercial product that costs money. It supports Visual Studio 2005, 2008, and also machines without Visual Studio at all (uses the Visual Studio Shell).

To read more about it, try it, or buy it, visit http://www.sapphiresteel.com/Ruby-In-Steel-Developer-Overview.

NetBeans

NetBeans is a free, open source IDE that supports several programming languages, along with the Ruby language. It is Java-based and can run on all operating systems that run Java applications.

For Ruby, you get code completion, naming convention warnings, a convenient project tree, and Ruby on Rails support.

Figure 4.8 shows the NetBeans window with Ruby code inside.

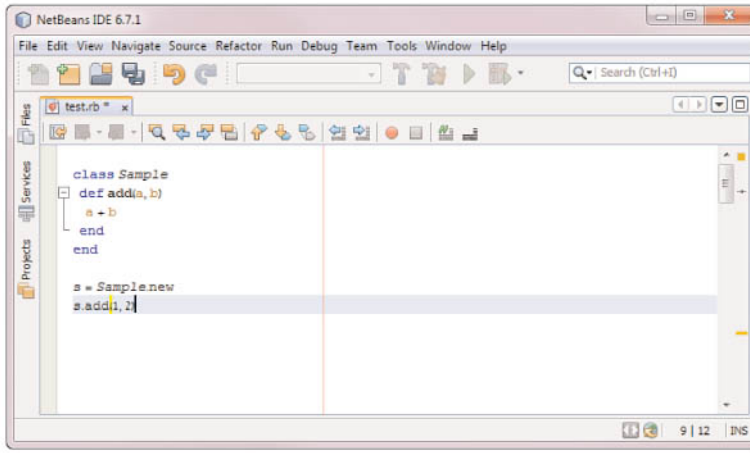

FIGURE 4.8 NetBeans IDE screenshot with Ruby code.

Unfortunately, NetBeans in its current version 6.7 doesn't support IronRuby. Therefore, you cannot run or debug IronRuby code directly from NetBeans. You must run the IronRuby file from the command prompt using ir.exe.

The NetBeans team plans to add IronRuby to its supported Ruby platforms in one of its next versions.

To learn more about NetBeans and download it, visit http://www.netbeans.org.

RubyMine

RubyMine is a commercial Ruby IDE by JetBrains. This is one of the most advanced Ruby IDEs available and features project creation wizards, syntax highlighting, code tools (like a "surround with" function), intellisense, refactoring, and version control system integration.

Figure 4.9 shows the RubyMine interface and its intellisense pop-up.

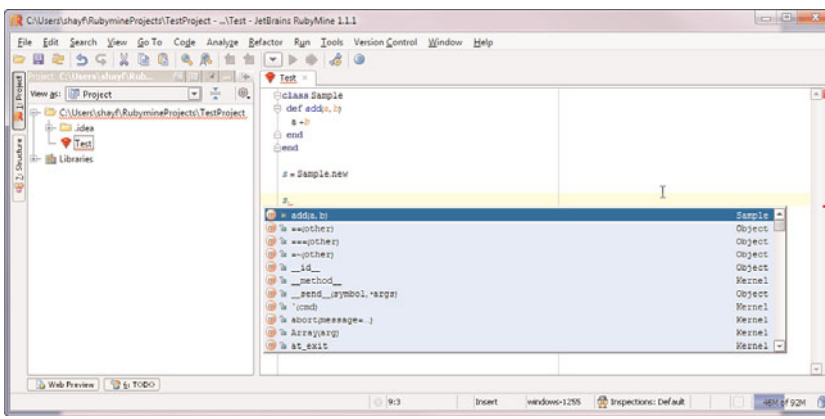

FIGURE 4.9 RubyMine IDE screenshot with intellisense.

RubyMine doesn't come with IronRuby as its Ruby interpreter, and you have to add it as one to execute files with the IronRuby interpreter directly from the interface.

Follow these steps to add IronRuby as a Ruby interpreter in RubyMine:

1. Go to File > Settings.
2. On the left, choose Ruby SDKs and Gems.
3. On the settings on the right, click the Add SDK button, which is located in the upper-right corner of the dialog.
4. On the file selector dialog that opens, navigate to <IronRuby installation folder>\Bin\ir.exe. Click OK after you select the file.
5. When you click OK on the settings form, RubyMine makes IronRuby its default interpreter.

Before you run a file, you need to modify something else. RubyMine uses configuration settings for each execution. In this configuration, the command-line arguments are sent to the interpreter. The default ones do not work with IronRuby, and you need to remove them. Follow these steps to do that:

1. Go to Run > Edit Configurations.
2. Click the Edit Defaults button, which is located in the lower-left corner of the dialog.
3. Choose Ruby on the left panel.
4. On the right, clear the text from the Ruby Arguments field.
5. Click OK on all open dialogs to save the changes.

Now you can work on and run files by using the IronRuby interpreter.

RubyMine is a commercial product that costs money. It works on Windows, Mac OS X, and Linux. You can read more about it, try it, and buy it at http://www.jetbrains.com/ruby.

Others

Along with these IDEs, a lot of other great IDEs are available. Some are appropriate for bigger applications, and some for smaller applications and scripts. Most of them offer simple code completion and syntax highlighting.

Like the others, they still cannot run IronRuby directly from their interface, so you have to switch to the command prompt and use ir.exe to run the Ruby file you've been working on.

Some of these IDEs are RadRails (http://www.aptana.com/radrails), SciTE (http://www.scintilla.org/SciTE.html), and Notepad++ (http://notepad-plus.sourceforge.net). Search the Internet for "Ruby IDEs" to find more Ruby IDEs.

The Power of IronRuby

As a finale to the IronRuby introduction, I want to leave you wanting more. Instead of just writing a simple Hello World application, I want you to see how great IronRuby is and how it can enhance your development work.

Ruby comes with very powerful built-in metaprogramming capabilities. One of its features is a built-in method called `method_missing` (read more about it in Chapter 6, "Ruby's Code-Containing Structures"). The `method_missing` method catches all calls to undefined methods and lets you handle them the way you want.

In Listing 4.1, by using `method_missing`, I implement a `Recorder` class that records calls to methods and plays them back on request. Notice how easy it is to accomplish this with Ruby. Even though you might not be familiar with all the syntax elements, this code is pretty simple and straightforward.

LISTING 4.1 A Recorder Class Implementation

```
class Recorder
  def initialize
    @calls = []
  end
  def method_missing(method, *args, &block)
    @calls << [method, args, block]
  end
  def playback(obj)
    @calls.each do ¦method, args, block¦
      obj.send method, *args, &block
    end
  end
end
```

Now that the `Recorder` class is ready, we can take advantage of it. Unlike regular Ruby code, we can take advantage of IronRuby capabilities and run the `Recorder` class with .NET code. Listing 4.2 takes advantage of the `Stack` class of the .NET Framework and runs recorded operations on it using our `Recorder` class.

LISTING 4.2 Using the Recorder Class on a CLR Object

```
recorder = Recorder.new
recorder.push 5.6
recorder.pop
recorder.push 1
recorder.push "IronRuby"

# Run the recorded calls on the CLR Stack instance
```

```
stack = System::Collections::Stack.new
recorder.playback(stack)
recorder.playback(stack)

stack.each { |x| puts x }
# Prints "IronRuby
#         1
#         IronRuby
#         1"
```

Summary

After reading this chapter, you are ready to start developing with IronRuby on your computer. You have installed IronRuby and have been introduced to the tools it comes with. This chapter also covered a few options for IronRuby development environments. At the end of the chapter, you saw an IronRuby sample that gave you a taste of the great power IronRuby holds inside.

We now delve further into the Ruby language and examine the fundamentals of IronRuby, and then you learn how to use IronRuby with the different frameworks of Ruby and .NET, all on the road to becoming an IronRuby master.

Index

SYMBOLS

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V