firstPress™

Available as a PDF Electronic Book or Print On Demand

NetBeans™ Ruby and Rails IDE with JRuby

Chris Kutler and Brian Leonard

Apress®
THE EXPERT'S VOICE™

Contents

Chapter 1: Installing NetBeans IDE with Ruby Support 1

Downloading the IDE *1*

Installing the Java Software Development Kit *2*

Installing the IDE *3*

Adding Ruby to an Existing NetBeans Installation *4*

Summary *6*

Chapter 2: Trying Out the IDE 7

Creating a Ruby Project *7*

Creating a Rails Project *11*

Summary *16*

Chapter 3: Configuring Your Environment 17

Registering Ruby Installations *17*

Managing Gems *20*

Using the Ruby Gems Manager 21
Gems Repository Permissions 22
Using the Gem Tool from the Command Line 23
Installing Gems That Have Native Extensions 25

Adding Servers *26*

Accessing Databases from the IDE *27*

Creating Databases from the IDE 27
Registering Existing Databases with the IDE 28

Working with Databases from the IDE 30

Summary ***32***

Chapter 4: Working with Ruby Projects 33

Setting Up Ruby Projects ***33***

Using the Project and Files Windows ***36***

Adding Files ***38***

Running Ruby Applications ***40***

Running Rake Tasks ***43***

Using Interactive Ruby (IRB) ***44***

Summary ***46***

Chapter 5: Working with Rails Projects 47

Setting Up Rails Projects ***47***

Working with Databases ***51***

Setting Database Configurations 51
Using Databases with JRuby 55
Creating Databases 57
Creating and Changing Tables 58

Working with Generators ***60***

Adding Files ***62***

Running Rake Tasks ***62***

Adding Plugins ***63***

Running Applications ***64***

Using the Rails Console ***67***

Summary ***68***

Chapter 6: Editing Files 69

Live Code Assistance..69

About Semantic Coloring..72

Using Code Completion..74

Using Live Code Templates..77

About Hints and Quick Fixes..79

Refactoring Code..81

Navigating Code..86

Navigating Within a File 86
Navigating to Members..88
Navigating to Files 89

Viewing RDoc..96

Working with Task Lists..98

Viewing Recent Changes..99

Keyboard Shortcuts..100

Summary..102

Chapter 7: Testing and Debugging Projects 105

Creating Tests..105

Creating Unit Tests..106
Adding Fixtures..109
Creating RSpec Tests 109

Running Tests and Specs..112

Using Autotest..115

Debugging Applications..116

Running the Debugger..117

Working with Breakpoints 119
Using the Debugger Window 122

Using the Local History Feature *123*

Viewing Log Files *124*

Summary *125*

Chapter 8: Working with JRuby 127

Creating JRuby Projects *127*

Adding Java Libraries to JRuby Projects *129*

Using Java Classes and Methods in JRuby Projects *130*

Setting JRuby Runtime Properties *132*

Using JRuby in Java Projects *134*

Summary *137*

Chapter 9: Deploying Rails Applications 139

Creating WAR Files with the Warbler Plugin *139*

Installing the Warbler Plugin 139
Configuring Warbler 142
Creating the WAR File 143

Deploying to Application Servers *143*

Deploying to the GlassFish Application Server 143
Deploying to Tomcat 145

Summary *145*

Chapter 10: Customizing the IDE 146

Working with Editor Options *146*

Tuning the Code Completion Pop-Up 146
Setting Formatting Options 148

Customizing Live Code Templates 150

Adjusting Hints.. 152
Creating Code Shortcuts.. 153

Changing Fonts and Colors 155

Modifying Keyboard Shortcuts... 158

Modifying File Templates... 160

Editing a Template 161
Using Template Properties 162
Adding Licenses to a Template 164

Setting Task List Patterns 165

Working with Nodes, Files, and Folders.. 166

Displaying the Physical File Structure in the Projects Window 167
Adding Ruby Script File Extensions 168
Changing the Location of the Ruby Source Files Folder 168
Editing Hidden Files in the IDE 169

Tweaking Under the Hood.. 169

IDE Options.. 170
Editor Options 172

Troubleshooting the IDE 173

Summary... 174

Chapter 1: Installing NetBeans IDE with Ruby Support

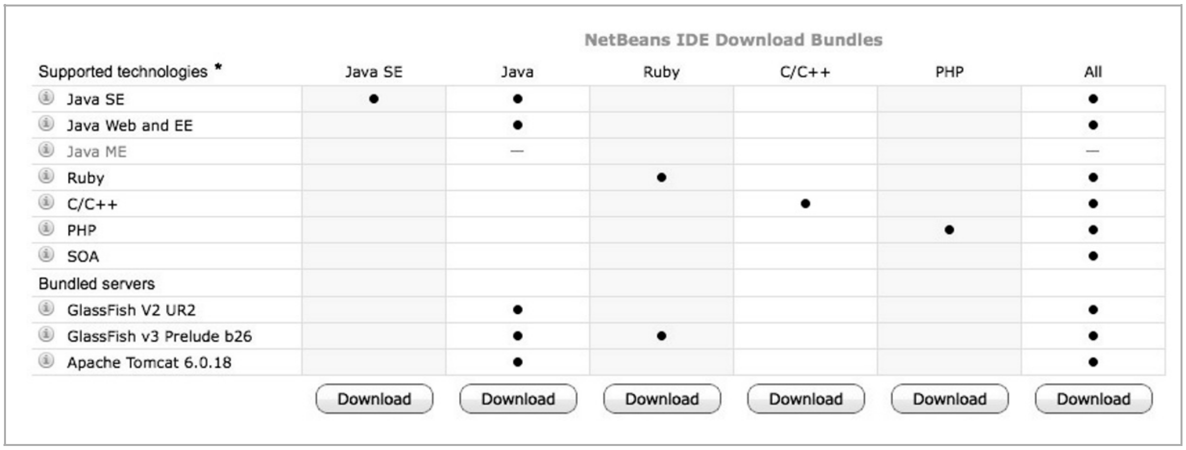
Installing the NetBeans IDE is fairly easy. Depending on your operating system, it may be as simple as running a command to fetch the IDE from a repository. Otherwise, you simply download the installation package from the NetBeans web site, ensure that you have the necessary Java Development Kit (JDK) software on your system, and run the IDE's installer. If you already have a 6.5 version of the IDE and want to add Ruby support, skip to the section titled "Adding Ruby to an Existing NetBeans Installation," later in this chapter.

Downloading the IDE

To get the NetBeans software, go to the NetBeans download page at `http://www.netbeans.org/downloads` and click the Download NetBeans IDE button. Choose a language and a platform, then click the Download button for either the Ruby bundle or the All bundle (see Figure 1-1). Because the Ruby download has a small footprint, it is the best option to choose if you are only doing Ruby programming. If you plan to use the IDE to develop programs with the other supported languages, the All download bundle is the better choice. You can also choose a different download, such as the Java bundle, and add Ruby support after you install the download. Alternatively, look farther down the web page for the link to the MySQL GlassFish Bundle, which adds the MySQL database server to the download.

Be sure to install version 6.5 or later. Several of the features described in this book do not exist in earlier versions.

Figure 1-1. The NetBeans Download Page



	NetBeans IDE Download Bundles					
Supported technologies *	Java SE	Java	Ruby	C/C++	PHP	All
Java SE	●	●				●
Java Web and EE		●				●
Java ME		—				—
Ruby			●			●
C/C++				●		●
PHP					●	●
SOA						●
Bundled servers						
GlassFish V2 UR2		●				●
GlassFish v3 Prelude b26		●	●			●
Apache Tomcat 6.0.18		●				●
	Download	Download	Download	Download	Download	Download

Tip You can download and install the IDE and its JDK software dependencies in a single step on an OpenSolaris system using the Package Manager GUI, or from a terminal window using the following command: `pkg install netbeans`. For Linux systems, you might want to check the Linux repositories to see if they offer NetBeans IDE 6.5 packages.

Installing the Java Software Development Kit

For most of the bundles you must have JDK software on your system before installing the IDE. The NetBeans download page has a link to the JDK download page as well as to a NetBeans IDE download that includes the JDK. On Ubuntu, you can install the JDK 6 package from Multiverse. If you have Ubuntu 8.04 or later you can also use the Open JDK 6 packages from the Universe repository.

Tip If you install the Ruby-only version of the IDE and will not use any of the IDE's Java features, you can install the Java Runtime Environment (JRE) instead of the JDK. However, if you do not have the JDK you will not be able to use the Warbler plugin to generate WAR files. The JDK includes the JAR command, which is used by Warbler.

The installation instructions on the download page provide steps for the various operating systems. For example, on Windows you run the JDK installer.

Although you can install the IDE using JDK 5.0 software, installing with the JDK 6 release or later is recommended. For example, the LCD subpixel antialiasing changes in version 6 greatly improve the way fonts are displayed in the IDE on Windows. If you are on Ubuntu you should install, at a minimum, version 6 update 10 of the JDK software due to Abstract Window Toolkit (AWT) issues with the Beryl and Compiz desktops.

Tip If you are not sure of your JDK version, execute the following in a command window: `java-version`.

Installing the IDE

When you have downloaded the NetBeans installer and have the necessary JDK installed on your system, you are ready to install the IDE. The method for installing depends on your operating system, as described here:

- **Windows**: Double-click the file that has the `.exe` extension.
- **Solaris, OpenSolaris, and Linux platforms (assuming you haven't installed from a repository)**: The installer file has an `.sh` extension. Before you run the installer file, execute the following command to make the file

executable: `chmod +x ./<installer-file-name>`. On Ubuntu, make sure your JDK release is the 6 update 10 release or later. Otherwise you could get a blank screen.

- **Mac OS X**: The installer file has a `.dmg` extension. After you run the installer the panel that opens displays a package icon with an `.mpkg` extension. Click this icon to start the installation wizard.

When the installer displays the JDK option, make sure the installer is using the JDK 6 release or later. If the installer cannot locate the JDK installation you want to use, try starting the installer again, but this time pass `--javahome <path-to-jdk>` to the installer.

The Ruby, Java, and All bundles include the GlassFish V3 server. The default username is anonymous and the password is blank. The Java and All bundles also include the GlassFish V2 server. The default username and password for the V2 server are admin and adminadmin.

The download instructions on the NetBeans download page provide more detailed instructions for each operating system.

Tip If you upgrade the JDK after you install the IDE, you might want to open the `*netbeans-install-dir*/etc/netbeans.conf` file and edit the `netbeans_jdkhome` property to point the new installation.

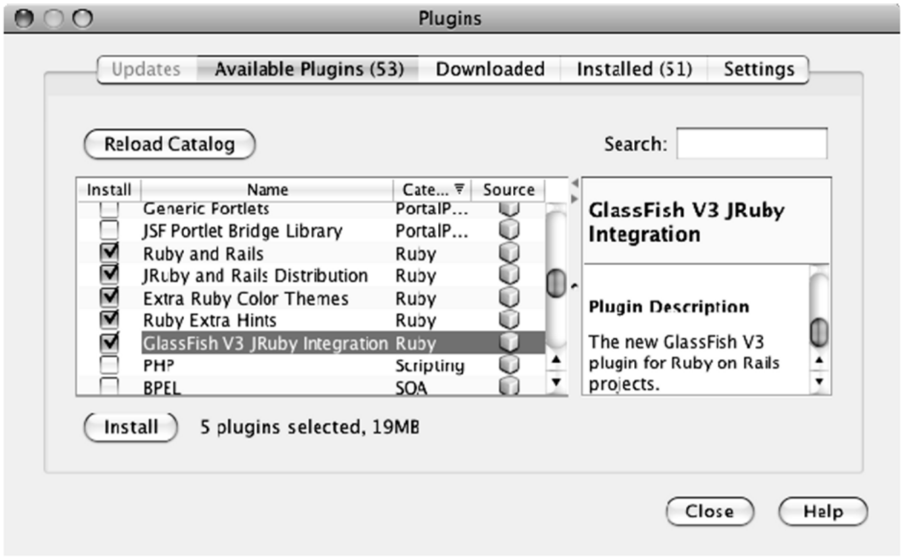
Adding Ruby to an Existing NetBeans Installation

If you have a NetBeans IDE 6.5 installation without Ruby support (that is, there is no Ruby category when you go to create a new project), you can easily add Ruby support to the IDE by doing the following:

1. If your network uses a proxy, choose Tools ➤ Options from the main menu, select Manual Proxy Settings, type the HTTP Proxy and Port for your proxy, and click OK.

2. Open the Plugins manager by choosing Tools ➤ Plugins from the main menu.
3. Click the Available Plugins tab and click the Category header to sort the list.
4. Select the check box next to the Ruby modules of interest (see Figure 1-2). At a minimum, select Ruby and Rails, which provides all of the NetBeans support for Ruby and Ruby on Rails (projects, editing, and so forth).
5. Unless you already have a Ruby or JRuby runtime installed, you will also want to select JRuby and Rails Distribution.
6. The other plugins in the Ruby category are optional, but there's really no reason not to install them as well, especially the GlassFish V3 JRuby Integration plugin.
7. Click the Install button to launch the Plugin Installer.
8. Accept the license and click Install to complete the installation.

Figure 1-2. Plugins Manager



Install	Name	Cate...	Source
☐	Generic Portlets	PortalP...	
☐	JSF Portlet Bridge Library	PortalP...	
✓	Ruby and Rails	Ruby	
✓	JRuby and Rails Distribution	Ruby	
✓	Extra Ruby Color Themes	Ruby	
✓	Ruby Extra Hints	Ruby	
✓	GlassFish V3 JRuby Integration	Ruby	
☐	PHP	Scripting	
☐	BPEL	SOA	

GlassFish V3 JRuby Integration

Plugin Description

The new GlassFish V3 plugin for Ruby on Rails projects.

5 plugins selected, 19MB

Summary

In this chapter, you learned how to obtain and install the NetBeans 6.5 IDE with Ruby support, as well as the JDK software. The installation process is fairly automated, regardless of whether you are developing on Windows, Mac, Solaris, or some flavor of Linux. In addition, the IDE enables you to install a Ruby-only version or a different configuration with added Ruby support.

Chapter 2: Trying Out the IDE

At this point you are probably anxious to give the IDE a test run. Since the NetBeans IDE provides everything you need to create Ruby and Rails applications out of the box, we thought we'd take you for a quick test drive before we dive into the details. As we go along through this chapter, we'll reference the chapters that go more in-depth on various topics.

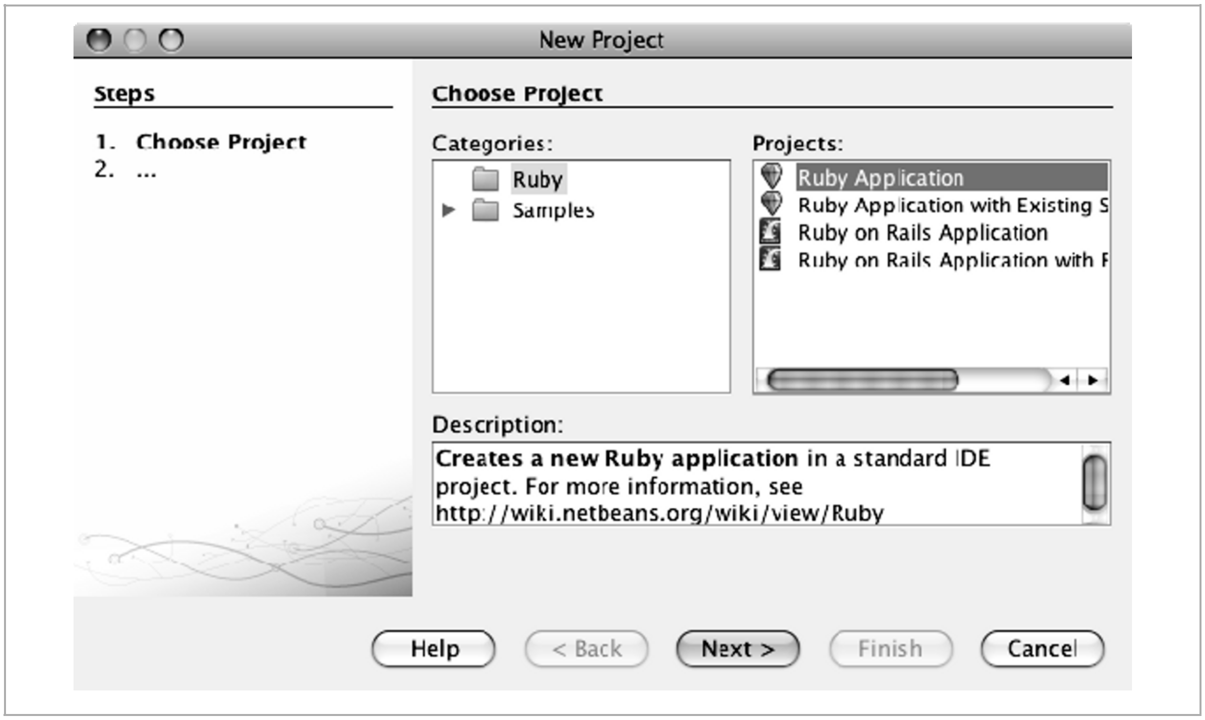
Creating a Ruby Project

Follow these steps to create a new Ruby project:

If you haven't done so already, start the IDE by clicking the NetBeans icon.

1. To create a project, select File ➤ New Project from the main menu. A New Project wizard appears from which you specify the type of project you want to create.
2. Select Ruby in the Categories list, select Ruby Application in the Projects list (see Figure 2-1), and click Next.
3. For this tour, use all the default settings. Note the assigned Project Name and Project Location and click Finish. You will learn more about using the New Project wizard for Ruby projects in Chapter 4.

Figure 2-1. Choosing the Project Type



New Project

Steps

1. Choose Project
2. ...

Choose Project

Categories:

- Ruby
- Samples

Projects:

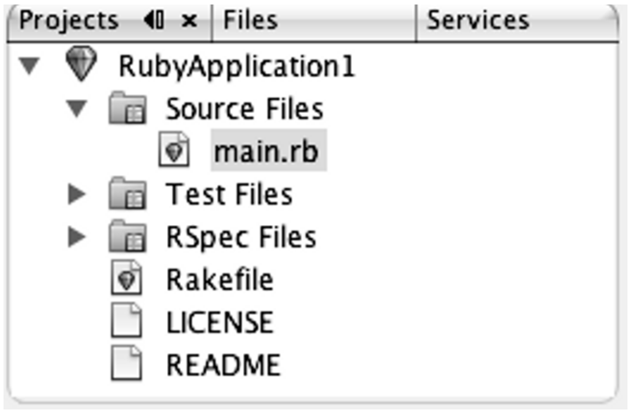
- Ruby Application
- Ruby Application with Existing S
- Ruby on Rails Application
- Ruby on Rails Application with F

Description:

Creates a new Ruby application in a standard IDE project. For more information, see http://wiki.netbeans.org/wiki/view/Ruby

4. On the left side of the IDE is a panel that contains tabs for the Projects, Files, and Services windows. You will work in the Projects window (see Figure 2-2). If you don't see the Projects window click the Projects tab to display it, or select Window ➤ Projects from the main menu. You should now see your Ruby project in the Projects window, with the main.rb node nested under the Source Files node.

Figure 2-2. Projects Window

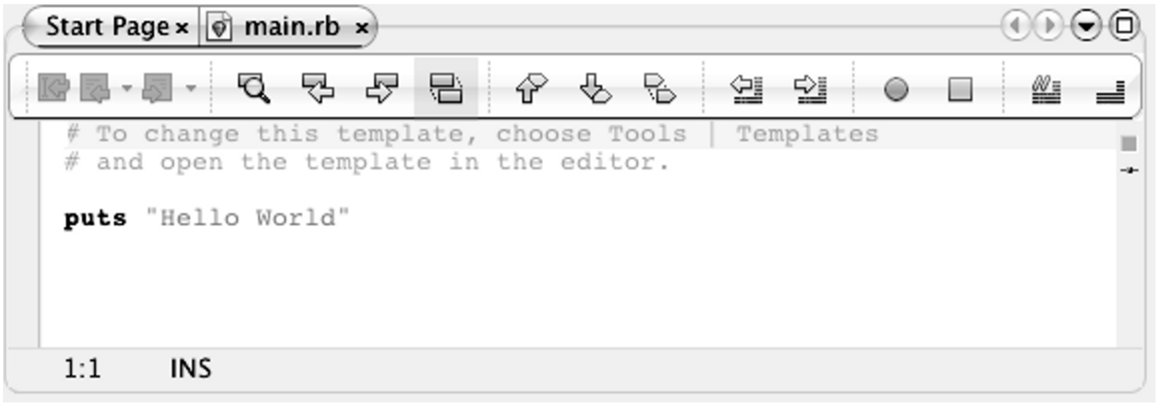


Note The Projects window shows a project's logical view; its pop-up menu contains actions you typically use to develop applications. The Files window shows the project's physical structure and has a pop-up menu for managing files.

---

5. In the main section of the IDE is the editor window. After the IDE creates the project it opens the `main.rb` file in this window (see Figure 2-3). The code in this file calls the `puts` method to display the string "Hello World." You will learn how to use the editor in Chapter 6.

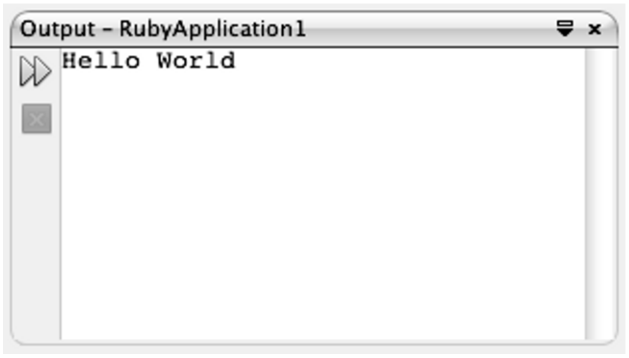
Figure 2-3. Editor Window



```
# To change this template, choose Tools | Templates
# and open the template in the editor.

puts "Hello World"
```

6. To run the application, right-click the project's node in the Projects window and choose Run from the pop-up menu. The IDE displays the output in the Output window at the bottom of the IDE (see Figure 2-4).

Figure 2-4. Output Window



```
Hello World
```

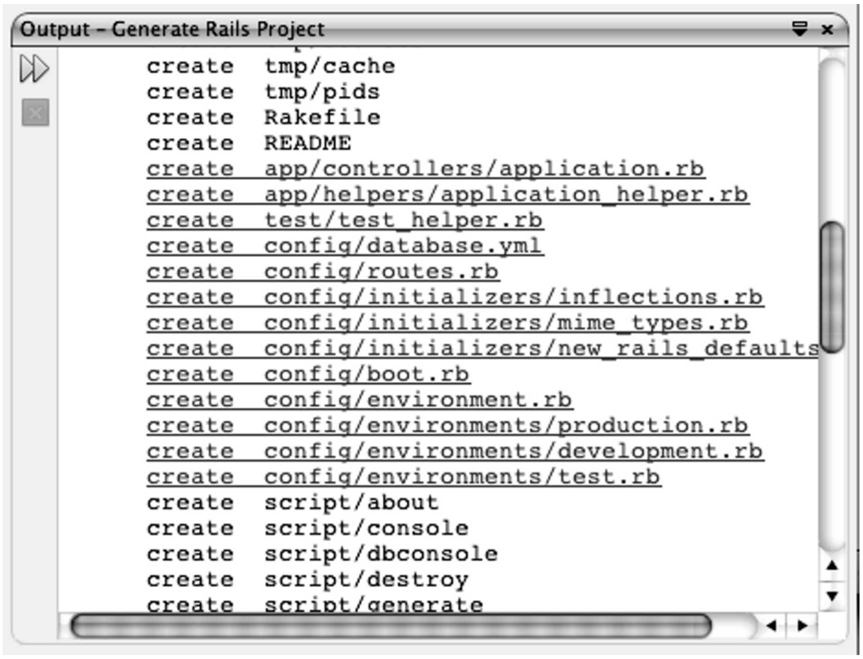
7. You can delete this project by right-clicking the project's node in the Projects window, then choosing Delete from the pop-up menu, selecting the Also Delete Sources check box, and clicking Yes.

Creating a Rails Project

Now follow these steps to create a Rails project:

1. Select File ➤ New Project from the main menu.
2. Select Ruby on Rails Application from the Projects list and click Next.
3. The New Project wizard for a Rails application has more steps, which you will learn about in Chapter 5, but we will skip them for this project. As you did when creating the Ruby project, make note of the Project Name and Project Location and click Finish. As part of the project creation process, the IDE runs the `rails` command to create a new Rails project and displays the output in the Output window (see Figure 2-5). The output shows the names of the created files as links. You can open any of these files in the editor by clicking the file's link.

Figure 2-5. Rails Command Results in the Output Window



```
      create  tmp/cache
      create  tmp/pids
      create  Rakefile
      create  README
      create  app/controllers/application.rb
      create  app/helpers/application_helper.rb
      create  test/test_helper.rb
      create  config/database.yml
      create  config/routes.rb
      create  config/initializers/inflections.rb
      create  config/initializers/mime_types.rb
      create  config/initializers/new_rails_defaults
      create  config/boot.rb
      create  config/environment.rb
      create  config/environments/production.rb
      create  config/environments/development.rb
      create  config/environments/test.rb
      create  script/about
      create  script/console
      create  script/dbconsole
      create  script/destroy
      create  script/generate
```

4. For this project you need to edit the `environment.rb` file to remove the Active Record framework from the environment configuration. Because you are not using a database in this application, you need to prevent Rails from trying to access the database specified in the `database.yml` file. Scroll through the Rails output until you see the line that says `create config/environment.rb` and then click the line to open the file in the editor window. Scroll to the following line in the source (around line 21):

```
# config.frameworks -= [ :active_record,
:active_resource,
:action_mailer ]
```

Tip To display line numbers, right-click in the left margin of the editor and choose Show Line Numbers from the pop-up menu.

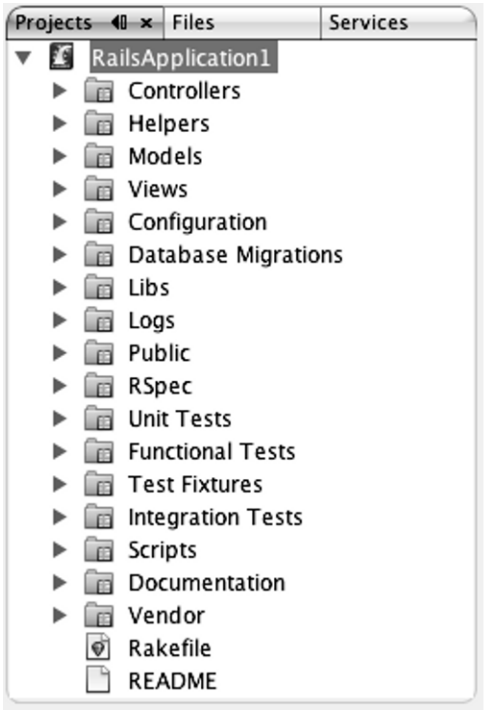
---

5. Click the `config.frameworks` line and click the Uncomment button on the editing tool bar to remove the # character from that line.
6. An asterisk (*) on the tab indicates the file has been modified. Click the Save All button on the main toolbar to save your changes.
7. If you are using a Rails 2.1 version, which is the case with the bundled JRuby software, you must also remove the Active Record references in `new_rails_defaults.rb`. Scroll through the Rails output until you see that line that says `create config/initializers/new_rails_defaults.rb`, then click that line to open the file in the editor window. Select each of the two `ActiveRecord::Base` settings and click the Comment button to comment out the lines, as shown here:

```
#ActiveRecord::Base.include_root_in_json = true
#ActiveRecord::Base.store_full_sti_class = true
```

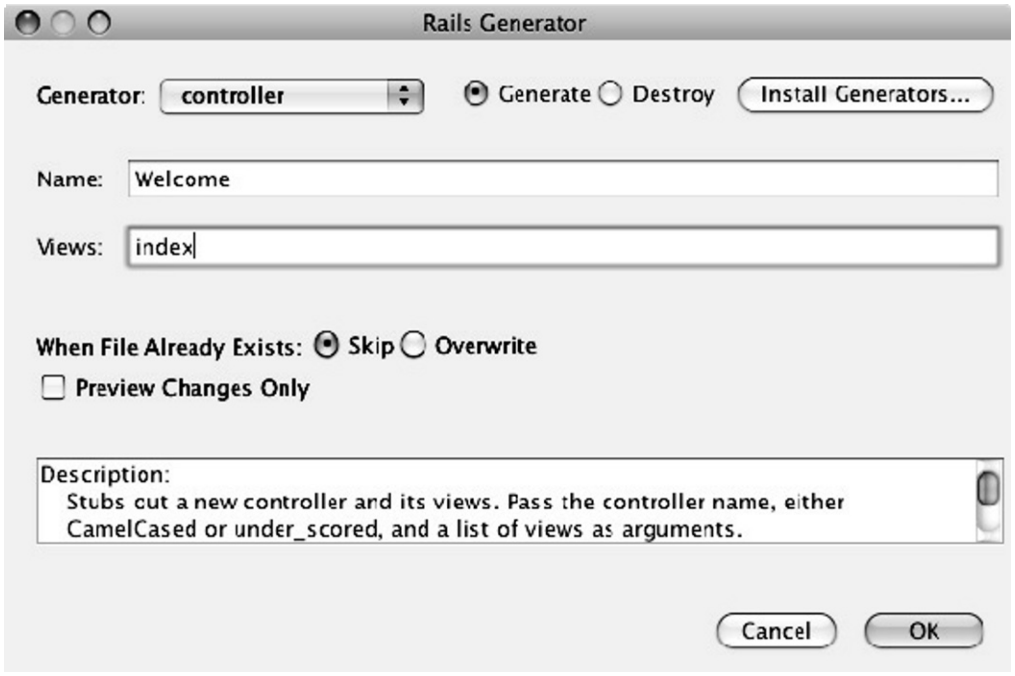
8. Click the Save All button on the main toolbar.
9. Now look at the Projects window, which shows a logical view of the Rails project, as shown in Figure 2-6.

Figure 2-6. Rails Project in the Projects Window



10. For this simple project you will create a controller and a view. In the Projects window, right-click the Controllers node and choose Generate from the pop-up menu. The IDE displays the Rails Generator dialog box with controller selected in the Generate drop-down list (see Figure 2-7). Set the Name to **Welcome**, set the Views to **index**, and click OK. The Rails Generator dialog box is a GUI interface to the Rails `generate` command. You will learn more about the Rails Generator in Chapter 5.

Figure 2-7. Rails Generator



Generator: controller [✓ Generate] Destroy

Name: Welcome

Views: index

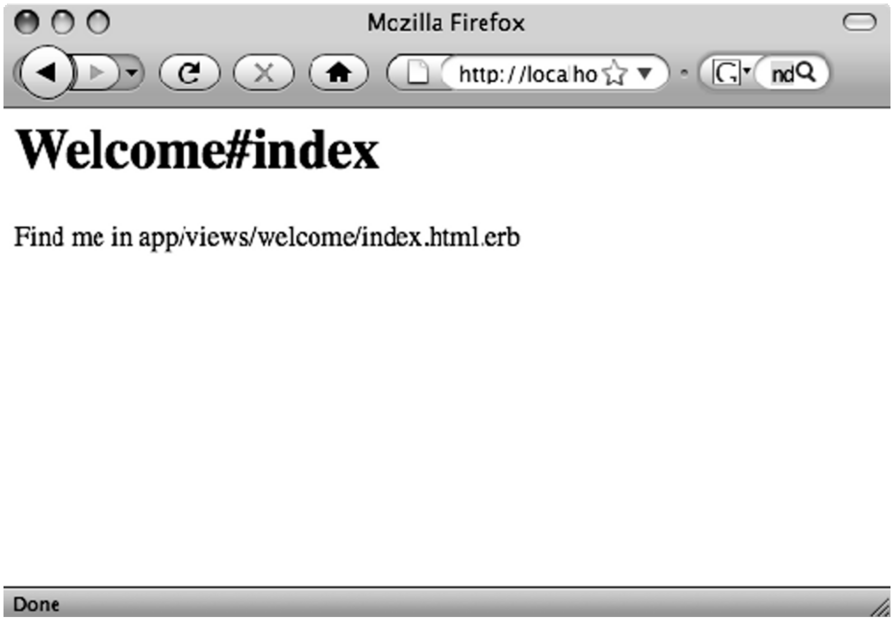
When File Already Exists: [✓ Skip] Overwrite

☐ Preview Changes Only

Description:
Stubs out a new controller and its views. Pass the controller name, either CamelCased or under_scored, and a list of views as arguments.

11. The IDE opens the `welcome_controller.rb` file in the editor. Right-click in the file's source and choose Run File from the pop-up menu. The IDE starts the server, runs the index action in the controller, and displays the index view in your browser. You should see a page that looks similar to Figure 2-8.

Figure 2-8. Index View in a Browser Window



Welcome#index

Find me in app/views/welcome/index.html.erb

Note If you get a database access error you might not have saved the changes you made to the `environment.rb` and `new_rails_defaults.rb` files. To resolve this problem click the Save All button on the main toolbar, then stop the server by clicking the X button on the left side of the server's tab in the Output window and try running the controller file again.

---

12. You can delete this project by right-clicking the project's node in the Projects window and choosing Delete from the pop-up menu. Then select the Also Delete Sources check box and click Yes.

Summary

This quick tour introduced some NetBeans basics. The workflow in the IDE is based on projects. You use the New Project wizard to create projects and you use the Projects window to manage your projects. The pop-up menu in the Projects window gives you access to most of the actions you perform on a project. The Output window displays the results of an action. The Files window shows the physical view of your files, and you can use the pop-up menu in this window to manage files. When you open a file, the IDE displays the file's content in the editor. The pop-up menu in an opened file gives you access to common editing tasks for that file's type.

Chapter 3: Configuring Your Environment

As you learned in the previous chapter, the NetBeans Ruby support works out of the box. You can immediately create a Ruby application or a Ruby on Rails application without having to install a Ruby interpreter, the Rails framework, or a Rails server. This is because the Ruby package includes the JRuby platform, the Rails gem, and both the WEBrick server and the GlassFish V3 server.

If you are already working on Ruby projects you probably want to use your own tools. This chapter shows you how to configure the IDE so you can use different Ruby interpreters, gem repositories, and Rails servers.

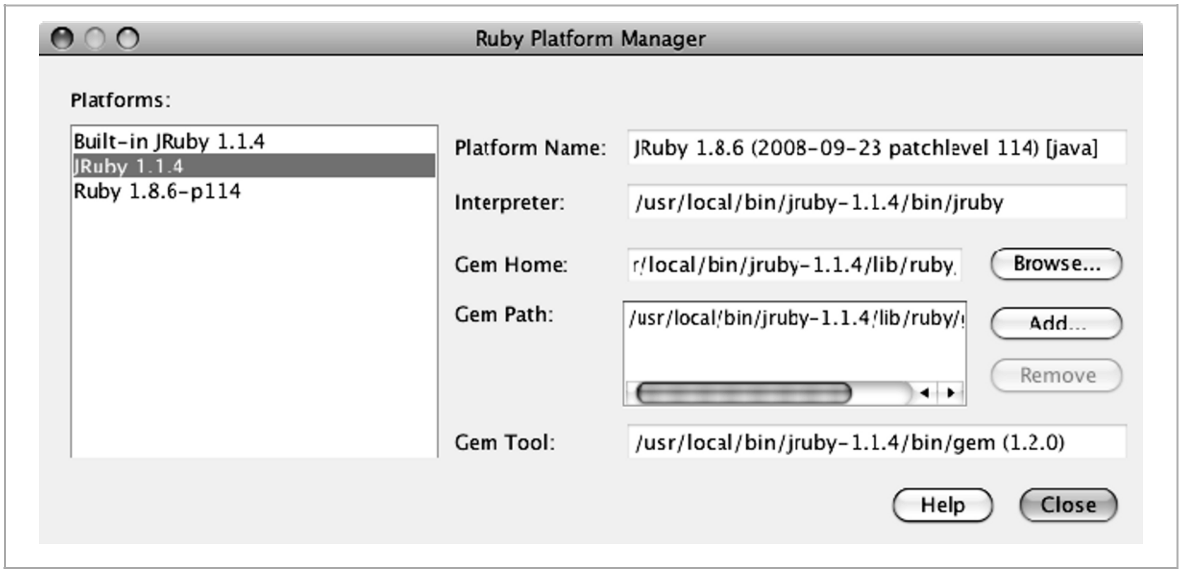
In addition, this chapter talks about registering your databases with the IDE. You don't need to register a database in order to use it in a Rails project; however, the IDE provides many helpful database features that you might want to take advantage of.

Registering Ruby Installations

When you installed the IDE (or when you added Ruby support to an existing installation of the IDE), the IDE looked on your system for existing JRuby, MRI Ruby, and Rubinius interpreters and automatically registered them with the IDE.

To view and add to the list of registered interpreters, open the Ruby Platform Manager (see Figure 3-1) by choosing Tools ➤ Ruby Platforms from the main menu. If you install Ruby after installing the IDE, click the Autodetect Platforms button to add the new Ruby interpreter to the list. If the IDE doesn't find your installation, you can add it manually by clicking the Add Platform button. When the file browser appears, navigate to the installation's `bin` folder and select the Ruby interpreter (such as `ruby.exe` or `jruby.bat`).

Figure 3-1. Ruby Platform Manager

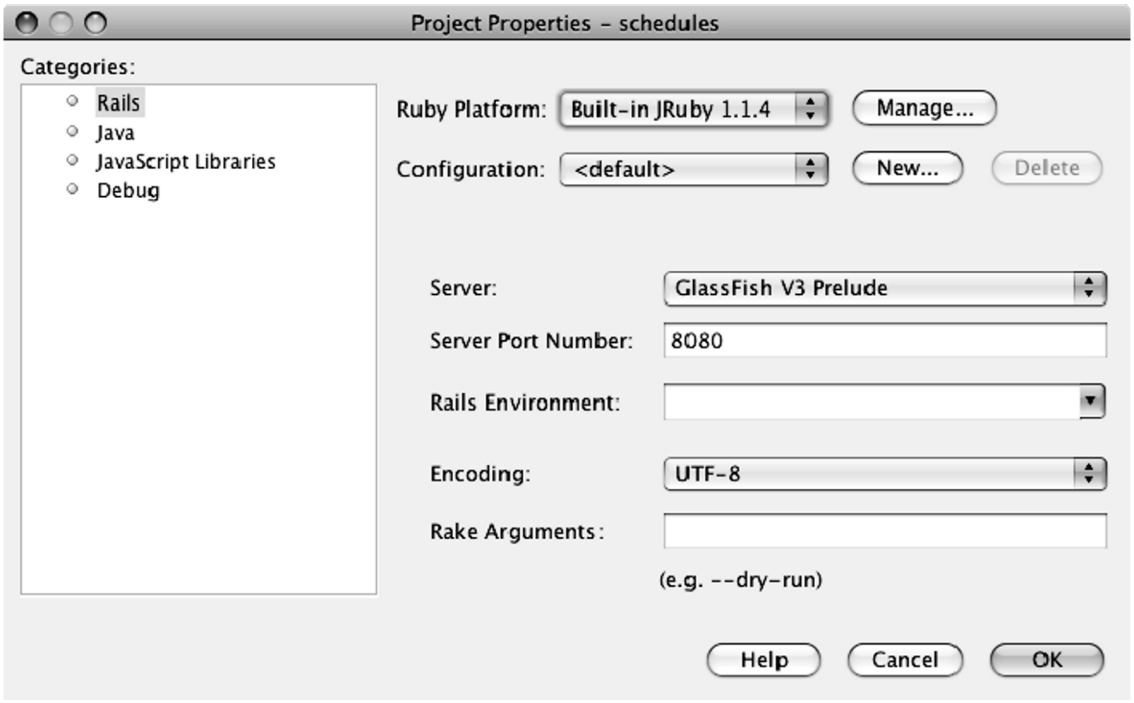


Platform Name: JRuby 1.8.6 (2008-09-23 patchlevel 114) [java]
Interpreter: /usr/local/bin/jruby-1.1.4/bin/jruby
Gem Home: r/local/bin/jruby-1.1.4/lib/ruby,
Gem Path: /usr/local/bin/jruby-1.1.4/lib/ruby/
Gem Tool: /usr/local/bin/jruby-1.1.4/bin/gem (1.2.0)

Tip Even though the IDE comes bundled with JRuby, you might want to download and install a separate distribution that is not located under the NetBeans installation folder. That way, if you update the IDE you will not need to change the location of the JRuby interpreter. Nor will you have to copy over or reinstall gems into the subfolders of the newer version of the IDE.

When you create a new Ruby project or Ruby on Rails project you can choose which of the registered platforms to use for the project (you will learn more about creating projects in Chapters 4 and 5). After you create a project you can easily change the project's interpreter by right-clicking the project's node, choosing Properties from the pop-up menu, and choosing a new Ruby Platform from the drop-down list in the Project Properties dialog box (see Figure 3-2).

Figure 3-2. Project Properties Dialog Box


Project Properties – schedules

Categories:
- Rails
- Java
- JavaScript Libraries
- Debug

Ruby Platform: Built-in JRuby 1.1.4

Configuration: <default>

Server: GlassFish V3 Prelude

Server Port Number: 8080

Rails Environment:

Encoding: UTF-8

Rake Arguments:

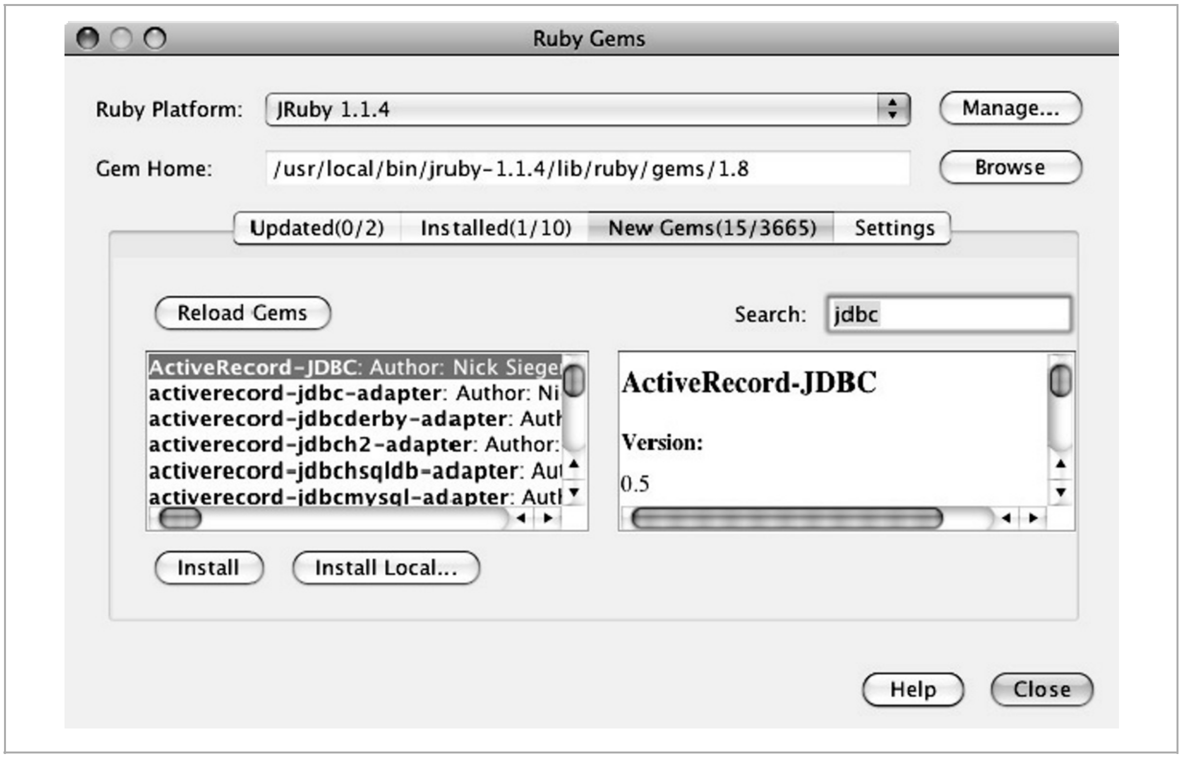
(e.g. --dry-run)

Note When you change a project's interpreter you probably need to make other changes to the project to make the application run with the new interpreter. For example, the new interpreter's gem repository might not contain all the gems the application depends on. If the new interpreter is using a different Rails version you will likely have to run the `rails:update` task or edit the `RAILS_GEM_VERSION` in the `environment.rb` file. When switching from a Ruby interpreter to a JRuby interpreter you might need to add a statement to the `environment.rb` file to require the JDBC Adapter. Also, when switching from a Ruby interpreter to a JRuby interpreter, or the other way around, you might need to change the configurations in the `database.yml` file. See Chapter 5 for more information.

Managing Gems

The IDE provides the Ruby Gems Manager, which is a GUI wrapper for the Ruby `gem` command (see Figure 3-3). This manager comes in handy for plain-vanilla gem management tasks such as viewing the repository contents and installing and updating the latest version of a gem.

Figure 3-3. Ruby Gems Manager


Ruby Gems

Ruby Platform: JRuby 1.1.4

Gem Home: /usr/local/bin/jruby-1.1.4/lib/ruby/gems/1.8

Search: jdbc

ActiveRecord-JDBC: Author: Nick Siege
activerecord-jdbc-adapter: Author: Ni
activerecord-jdbcderby-adapter: Auth
activerecord-jdbch2-adapter: Author:
activerecord-jdbchsqldb-adapter: Au
activerecord-jdbcmysql-adapter: Auth

ActiveRecord-JDBC

Version:

0.5

Tip Two JRuby gems you might want to install right away are the JRuby OpenSSL gem and the Active Record JDBC Adapter for your database server.

You cannot use this GUI to perform some repository actions, such as uninstalling a specific version of a gem. In these cases you must use the `gem` command in a terminal window, as described later in this chapter in "Using the Gem Tool from the Command Line."

Note If the Gem Home setting says `RubyGems are not installed for this platform`, see the instructions at `http://www.rubygems.org/read/chapter/3` to learn how to install the gem tool for that platform. Also, if the Ruby Gems Manager displays a warning that you have an old version of RubyGems, see the instructions at `http://www.rubygems.org/read/chapter/3#page14`.

Using the Ruby Gems Manager

To access the Ruby Gems Manager, select Ruby Gems from the Tools menu. Next select the Ruby Platform for the repository that you want to manage. In the Gem Home text box the IDE displays the path to the repository that the gem tool uses for adding and maintaining gems for the selected platform.

The Ruby Gems Manager has four tabs, as shown earlier in Figure 3-3:

Updated. This tab lists available updates for the gems that are installed in your repository. The list shows the version number of the installed gem and the latest version that is available. When you select a gem the IDE displays its details in the right panel, including the description. You can select a gem and click Update, or you can click Update All to bring all your installed gems up to date.

Installed. This tab shows you the gems that are installed in the repository, with a button to uninstall the selected gem.

New Gems. This tab allows you to add new gems to a Ruby installation. This is best done using the Search text box, which is also available for the Updated and Installed tabs but is most useful here, where more than 3,600 gems (and growing rapidly) appear in the list. After you type in the Search textbox, the total in the tab updates to indicate how many gems were found that match your search criteria, as shown in Figure 3-3. By default, only the most recent versions are listed. You can change this in the Settings tab. To install a gem, select the gem, click Install, and then, if prompted, select the version. If you have already downloaded the gem, click Install Local and then navigate to and select the downloaded gem.

Settings. This tab allows you to configure proxies. You can also use this tab to specify whether to fetch all gem versions and whether to fetch detailed gem descriptions. Turning off these features speeds up gem fetching.

Tip If you get errors using the Ruby Gems Manager, try the equivalent gem command from the command line. You might get better diagnostic messages when you use the gem tool directly. See "Using the Gem Tool from the Command Line" later in this chapter.

Gems Repository Permissions

On most systems other than Windows, the folder that contains the gem repository is often privileged. When necessary, the IDE invokes the underlying `gem` command using `gksu` or `kdesu`, if it finds it. If the IDE reports permission problems with the repository and you are not able to make the repository writable for the IDE, you will have to manage your gems from the command line. Alternatively, you can create a folder with write permissions and make that folder the Gem Home by following these steps:

1. Choose Tools ➤ Ruby Platforms or click the Manage button in the Ruby Gems Manager to open the Ruby Platform.
2. Click Browse and navigate to where you want to put the gems repository.
3. Click the New Folder button.

4. Name the new folder, select it, and click Open.
5. The IDE asks if you want to set up a new repository. Answer OK.

Note Vista users might encounter User Account Control issues. If this happens see `http://wiki.netbeans.org/FaqInstallPluginVista` for help.

Using the Gem Tool from the Command Line

Sometimes you will not be able to use the Ruby Gems Manager to install or update a gem. When that happens, open a terminal window and invoke the appropriate command from the command line.

Tip Before running gem commands from the command line you might want to verify that either `GEM_HOME` is not set, or that it points to the desired repository.

Using the Gem Command with JRuby

To invoke the Gem Tool for your JRuby gem repository you must first add the JRuby installation's `bin` directory to your `$PATH` (`%PATH%` on Windows) environment variable. Otherwise you must specify the full path to the `jruby` script. To find the path to the `jruby` script, open the Ruby Platform Manager, select the JRuby platform, and look at the Interpreter setting.

Note If you have Ruby installations on your system, the JRuby documentation recommends that you add the path to the JRuby `bin` directory to the end of the `PATH` environmental variable.

To invoke the Gem Tool for JRuby, use `jruby -S gem` *`arguments`*, as shown here:

```
jruby -S gem install -v=2.0.2 rails
```

The `-S` parameter tells JRuby to look for the script in its `bin` directory.

For more information see `http://wiki.jruby.org/wiki/Getting_Started`.

Note With earlier versions of JRuby you might get an out-of-memory error when invoking the Gem Tool. If this happens use the `Xmx` parameter to increase the memory. Here is an example:

```
jruby -J-Xmx1024m -S gem arguments
```

Using the Gem Command with CRuby

As with JRuby, you need to make sure your CRuby (MRI Ruby or Rubinius) installation's `bin` directory is in the `PATH` environment variable. To invoke the Gem Tool for CRuby use `gem` *`arguments`*, as this example shows:

```
gem install -v=2.0.2 rails
```

For more information, see *Installing a Gem* at `http://docs.rubygems.org/read/chapter/10#page33`.

Installing Gems That Have Native Extensions

Some gems build native C libraries during the install process. These gems will not work in JRuby applications unless the gem also provides a Java equivalent to the native library. For example, because the Mongrel gem builds its native library in a platform-independent manner, it works OK with JRuby. The following table shows a few examples of replacements for gems with native C libraries.

NATIVE C GEM	REPLACEMENT
RMagic, ImageScience	ImageVoodoo
OpenSSL	JRuby-OpenSSL
Ruby/LDAP	JRuby/LDAP
json	json_pure

You can add gems that have native extensions to CRuby repositories on Linux, Solaris, OpenSolaris, and Mac operating systems, provided you have the necessary software to enable the gem tool to compile the native extensions.

Linux. You need the `build-essential` package as well as the `ruby<version>-dev` package, which contains the header files for building the extension modules for Ruby.

Mac. You need the Xcode developer environment (see `http://developer.apple.com/tools/xcode`), which can be found on the Mac OS installation DVD.

OpenSolaris and Solaris. You need `SUNWgcc`.

If you are on Windows, check to see if there is an `mswin` version of the gem, which has been precompiled.

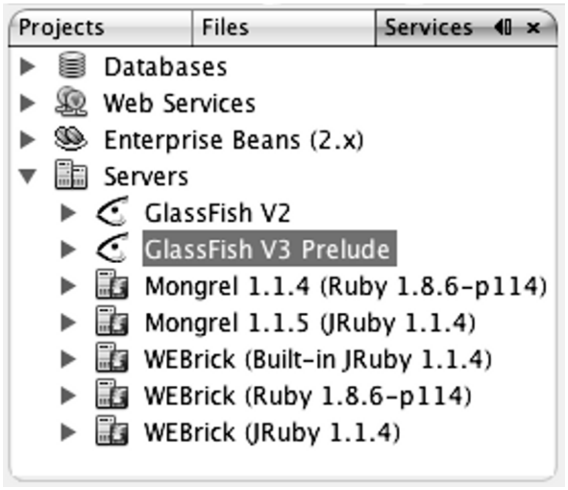
Tip If you are using both CRuby and JRuby you might want to set up a repository for the gems that you know work with both. Add the path to the shared repository to the Gem Path in the Ruby Platform Manager. To install a gem into the shared repository, open the Ruby Gems Manager and use the Gem Home Browse button to enter the shared repository's path. After you install the gem, remember to change the path back to its previous value.

Adding Servers

The WEBrick runtime server is bundled with Ruby. In addition, the IDE comes with the GlassFish V3 application server that runs both Rails and Java applications.

The IDE also provides integrated support for Mongrel, which you can install using the Ruby Gems Manager. When you install the Mongrel gem, the IDE adds it to the list of servers in the Services window (see Figure 3-4).

Figure 3-4. Servers Node in the Services Window



When you create a new project, you specify which server to use in step 2 of the New Project wizard. The default is GlassFish for JRuby projects and WEBrick for Ruby projects. You can change the project's server in the project's Properties window.

The NetBeans All and Java bundles include the GlassFish V2 application server. You can also download the GlassFish V2 application server separately, and then register it with the IDE by right-clicking the Servers node in the Services window and choosing Add Server. The GlassFish V2 application server is not fully integrated with Ruby support. However, you can use the Warbler plugin to create WAR files that you can deploy to the GlassFish V2 application server, as described in Chapter 9.

Note While you do not need the JDK to run the Ruby-only version of the IDE, you do need the JDK to use the Warbler plugin to create WAR files.

Accessing Databases from the IDE

The IDE makes it easy to work with any database for which JDBC 3.0 drivers are available. The IDE ships with drivers for the MySQL, PostgreSQL, and Java DB (also known as Derby) database servers, and you can obtain JDBC 3.0 database drivers for just about any other database.

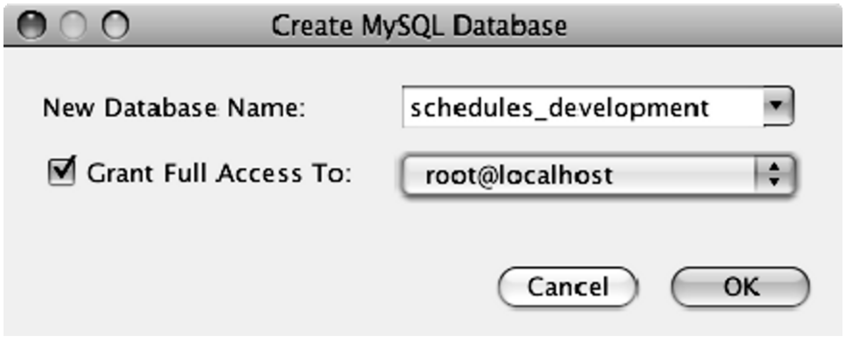
Creating Databases from the IDE

If your system has MySQL or Java DB, you can create databases directly from the IDE by completing the following steps. Java DB is bundled with the JDK 6 software and the GlassFish application server. You can install MySQL from mysql.com. There is also a NetBeans 6.5 distribution that bundles MySQL.

Follow these steps to create a MySQL or Java DB database:

1. Open the Services window and expand the Databases node.
2. Right-click the MySQL Server or Java DB node and choose Create Database.
3. Type the name of the database you want to create and click OK (see Figure 3-5). The IDE creates the database, registers it with the IDE, and creates a connection.

Figure 3-5. Create Database Dialog Box



New Database Name: schedules_development
☑ Grant Full Access To: root@localhost

With MySQL databases you can also create and register a database from the New Project wizard for a Rails Project, as shown in the following steps:

1. In step 3 of the wizard, click the Create DB button to open a Create MySQL Database dialog box (see Figure 3-5).
2. Type the name of the database you want to create and click OK . The IDE creates the database and registers it with the IDE.

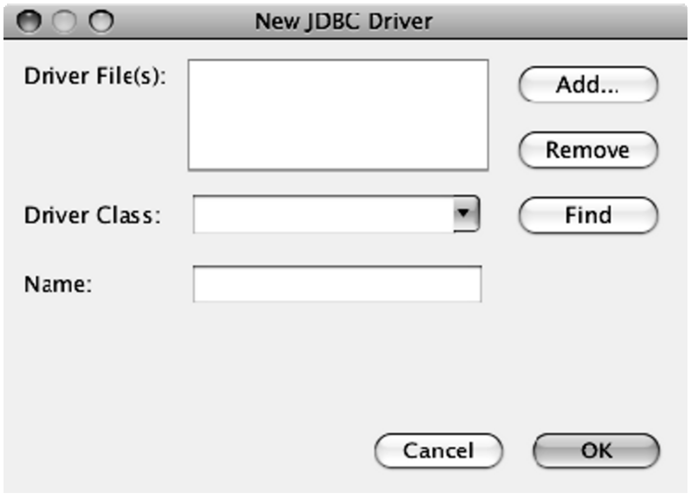
Registering Existing Databases with the IDE

As mentioned previously, you can register JDBC 3.0–compliant databases with the IDE.

The IDE ships with drivers for the MySQL, PostgreSQL, and Java DB database servers. If you are working with a different database server you first need to make the server's JDBC 3.0 driver available to the IDE, as shown in the following steps:

1. Expand the Databases node in the Services window, right-click Drivers, and choose New Driver to open the New JDBC Driver dialog box (see Figure 3-6).

Figure 3-6. New JDBC Driver Dialog Box



2. Click Add, navigate to and select the driver's JAR file, and click Open.
3. When possible, the IDE discovers and enters the values for Driver Class and Name. If not, you need to type these in.
4. Click OK.

Once the driver is added you can easily register your database with the IDE.

To register an existing database, follow these steps:

1. Right-click the Databases Node in the Services window and choose New Connection. The IDE opens the New Database Connection dialog box (see Figure 3-5).

2. Choose the Name of the database driver and provide the requested information.
3. Click OK. The IDE creates the database and registers it with the IDE.

Working with Databases from the IDE

The IDE provides a Database Explorer for creating tables, populating tables with data, and running SQL queries on registered databases. You must connect to the registered database before you can use the Database Explorer. In the Services window, expand the Databases node, right-click the node for your database, and choose Connect. After you establish a connection you can expand the database's node and perform the following tasks:

Create Database Tables. Right-click the database's Tables node and choose Create Table. Add the desired columns and click OK (see Figure 3-7). You can also create tables using the SQL Editor, which is explained next.

Figure 3-7. Create Table Dialog Box

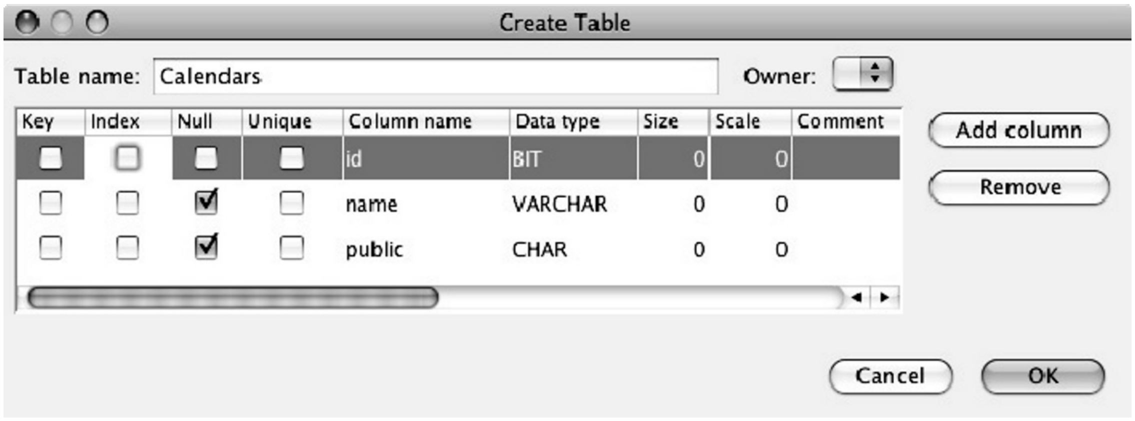
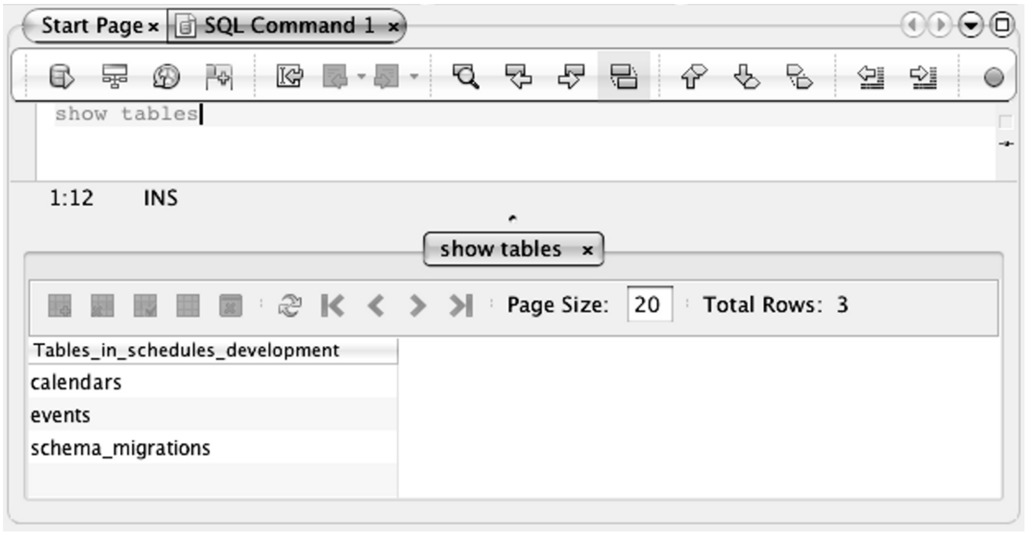


Table name: Calendars
Owner:

Key	Index	Null	Unique	Column name	Data type	Size	Scale	Comment
☐	☐	☐	☐	id	BIT	0	0	
☐	☐	✓	☐	name	VARCHAR	0	0	
☐	☐	✓	☐	public	CHAR	0	0	

Write and Execute Queries. Right-click the database's Tables node and choose Execute Command. Type the query, then click the Run SQL button in the editor's toolbar or right-click the query and choose Run Statement. Query results are shown in the SQL Editor output window (see Figure 3-8). If you are unsure of a schema, table, or column name, press Ctrl+Space (Cmd+Space and Cmd+\ on the Mac) to display the code completion pop-up. If you right-click and choose SQL History, you can view earlier queries.

Figure 3-8. SQL Editor Output Window



```
show tables
```

Page Size: 20   Total Rows: 3

Tables_in_schedules_development
calendars
events
schema_migrations

View Data, Insert Records, and Delete Records. Right-click one of the nodes under Tables and choose View Data. To add a record, click the Insert Record button in the data output tab. To delete a record, select it and click the Delete Record button.

Run SQL Scripts. From the main menu, choose File ➤ Open File. Navigate to the script file, select it, and then click Open. The script opens in the SQL Editor. Set the Connection to the desired database and click the Run SQL button.

Note You do not need to register a database server with the IDE to use its databases from Rails projects.

Summary

Although NetBeans' Ruby support works out of the box with the JRuby platform and the GlassFish server, you can easily configure the IDE to use different interpreters and servers. The Ruby Gems Manager enables you to quickly view and manage a platform's gem repository. However, for more complex actions, you might need to manage the repository from the command line.

Each project can be configured to use the platform and server of choice, and it is not too hard to switch a project from one platform to another or one server to another.

You can register a database with the IDE and take advantage of the Database Explorer's GUI tools to aid in working with your data.

Chapter 4: Working with Ruby Projects

In Chapter 2 you got a quick tour of working with Ruby projects. This chapter goes into more details about using the IDE to create and work on Ruby applications.

Setting Up Ruby Projects

The NetBeans IDE provides four Ruby project types:

- Ruby application
- Ruby application with existing sources
- Ruby on Rails application
- Ruby on Rails application with existing sources

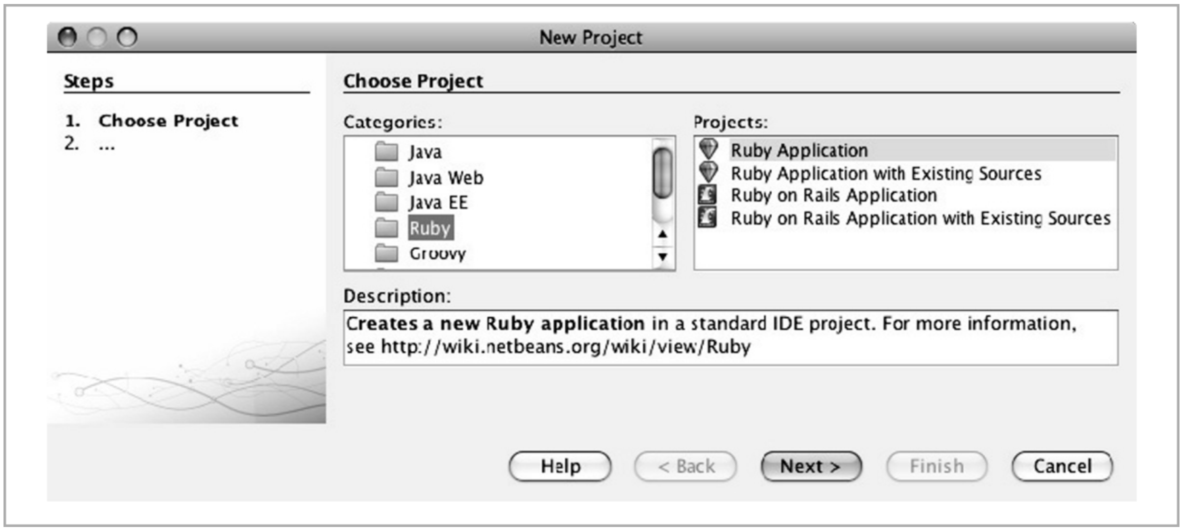
In this chapter we talk about the first two project types. We address the second two in the following chapter.

As the names imply, you can create a Ruby application either from scratch or from an existing Ruby application that you previously created outside of the IDE. If you already have Ruby sources lying around, you can easily and harmlessly work with them in the IDE by choosing to create a new Ruby application with existing sources. We say "harmlessly" because the IDE simply adds an `nbproject` directory to your project, from which it maintains metadata used by the IDE. Once you delete the folder, it's no longer a "NetBeans Project" (and this is exactly what NetBeans does when you delete a project from the IDE).

As with all NetBeans projects, you use the New Project wizard and follow these steps to create your projects:

1. Select File ➤ New Project to open the New Project wizard (see Figure 4-1).

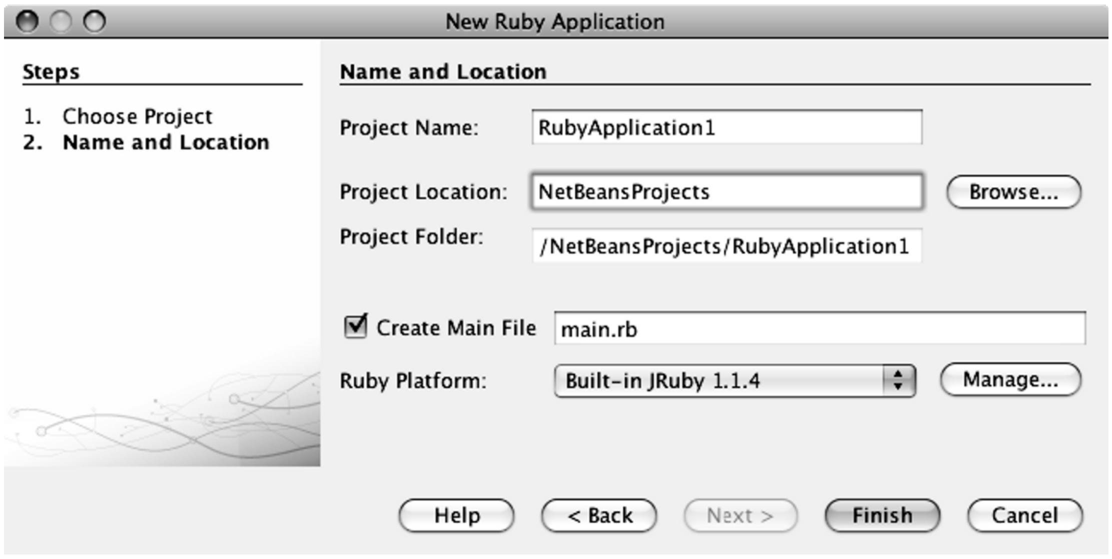
Figure 4-1. New Project Wizard



Creates a new **Ruby application** in a standard IDE project. For more information, see http://wiki.netbeans.org/wiki/view/Ruby

2. Select Ruby in the Categories list.
3. In the Projects list, select either Ruby Application or Ruby Application with Existing Sources.
4. Click Next to go to the Name and Location step (see Figures 4-2 and 4-3).

Figure 4-2. Name and Location Step for Creating a Ruby Application from Scratch



New Ruby Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: RubyApplication1

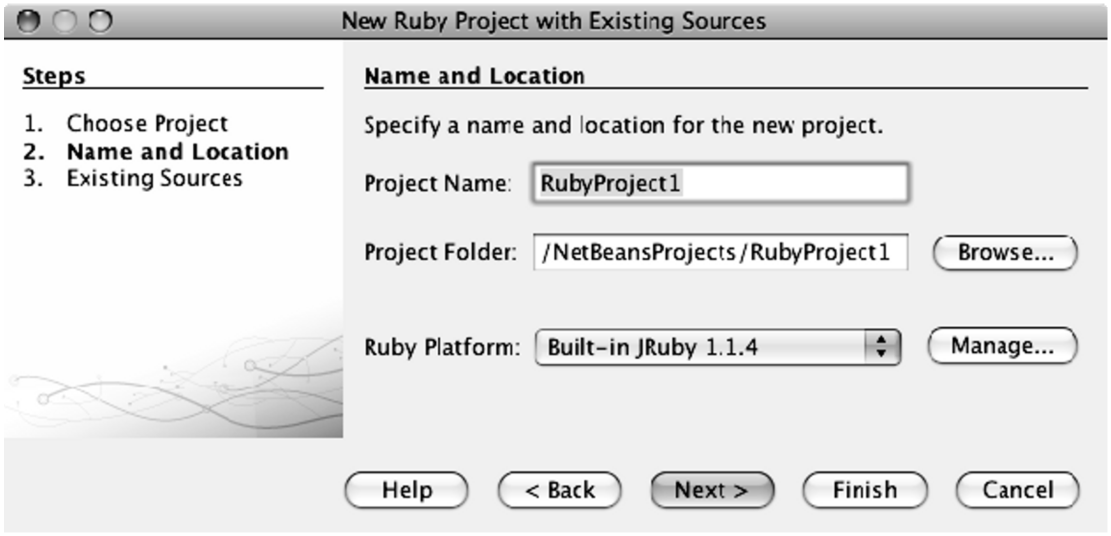
Project Location: NetBeansProjects

Project Folder: /NetBeansProjects/RubyApplication1

☑ Create Main File: main.rb

Ruby Platform: Built-in JRuby 1.1.4

Figure 4-3. Name and Location Step for Creating a Ruby Application from Existing Sources



New Ruby Project with Existing Sources

Steps

1. Choose Project
2. **Name and Location**
3. Existing Sources

Name and Location

Specify a name and location for the new project.

Project Name: RubyProject1

Project Folder: /NetBeansProjects/RubyProject1

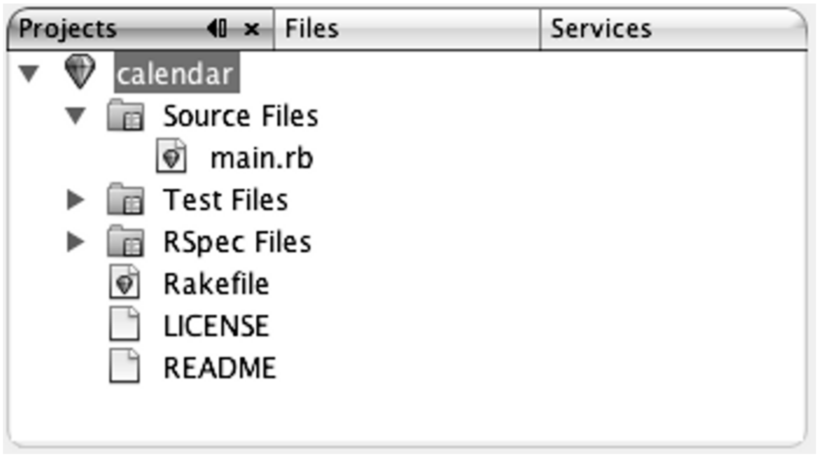
Ruby Platform: Built-in JRuby 1.1.4

5. Enter the project's name and location. For Ruby applications with existing sources, enter the path to the project, then click Next to specify the paths to the source (`lib`) and test folders.
6. If you are creating a project from scratch, specify whether to create a main file (the file that the IDE executes when you run the project). The IDE suggests the name **main.rb**, but you can change it.
7. Select the Ruby Platform that you want to use for this project. You can change this setting at any time using the Project Properties dialog box. If the desired platform is not in the drop-down list, click Manage to register the platform with the IDE.
8. Click Finish to create the project.

Using the Project and Files Windows

When you create a project from scratch the IDE creates a fully functional "Hello World" Ruby application, the `Rakefile` file, which we address in the "Running Rake Tasks" section, a `README` file, which you can use to provide basic documentation about your project, and a `LICENSE` file. The IDE also creates folders for your test files and RSpec files (see Figure 4-4).

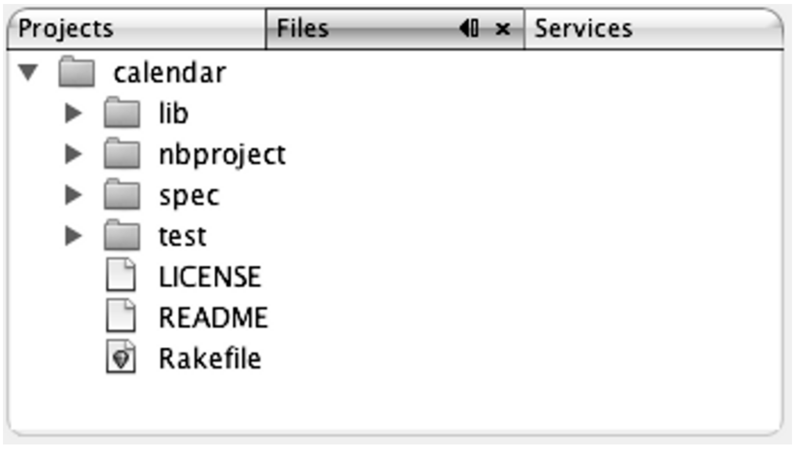
Figure 4-4. New Ruby Project in the Projects Window



When you build a new project from scratch, the Source Files node maps to the underlying `lib` directory, where Ruby source files generally reside (see Chapter 10 for instructions on how to change the location of the `Source Files` directory). The Test Files node and RSpec Files node map to the underlying `test` and `spec` directories, and are the intended locations for your Ruby tests (see Chapter 7 for more information about testing your Ruby projects).

If you want to see the physical layout of your project, click the Files tab (see Figure 4-5). As you learned in Chapter 2, the pop-up menu in the Projects window contains the actions that you typically use to develop Ruby applications, while the pop-up menu in the Files window has actions for managing the file system.

Figure 4-5. New Ruby Project in the Files Window



Tip See Chapter 10 to learn how to make the Projects window display a physical view of the files instead of a logical view.

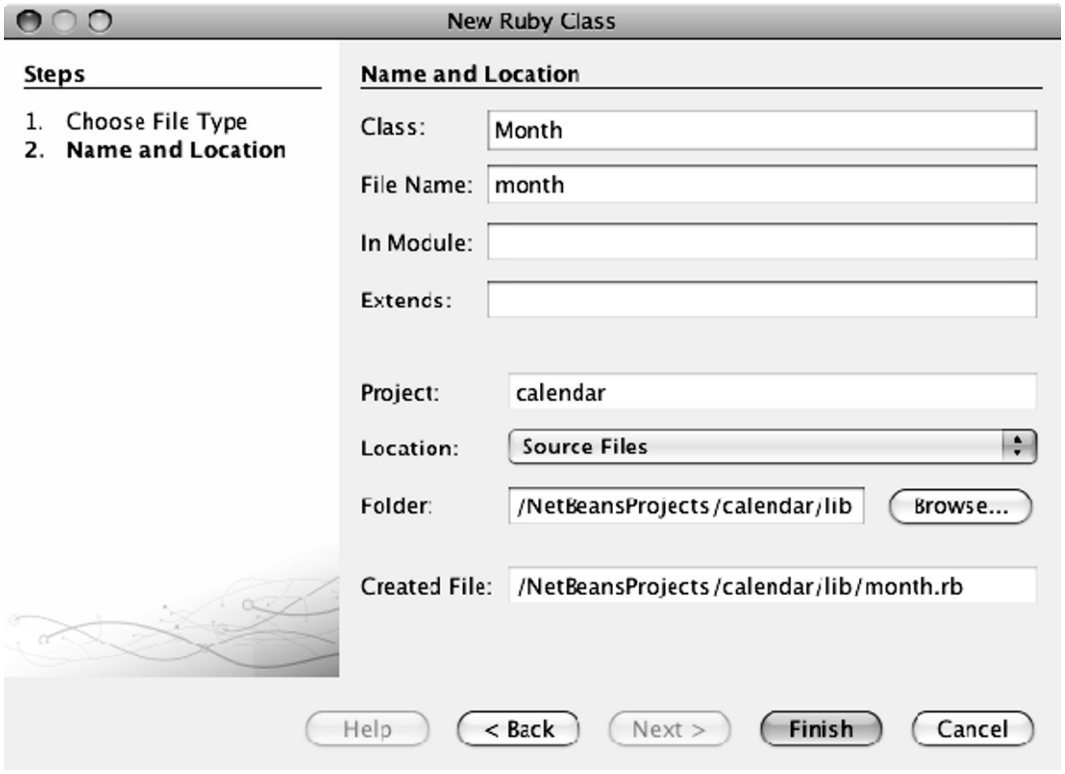
Adding Files

Now that you have created a project, you most likely need to add new files. The IDE provides several Ruby file templates (types) to help get you started, with Ruby File being the most generic, requiring simply a name and a location. A Ruby Module lets you specify an optional parent module, while a Ruby Class lets you specify an optional parent module and an optional parent class. You can use all three file types interchangeably; the Ruby Module and Ruby Class templates just help you set up the file structure a bit.

For example, to create a new class, complete the following steps:

1. In the Projects window, right-click one of the project's nodes, such as the Source Files node, and choose New ➤ Ruby Class to open the New Ruby Class dialog box (see Figure 4-6).

Figure 4-6. New Ruby Class Dialog Box

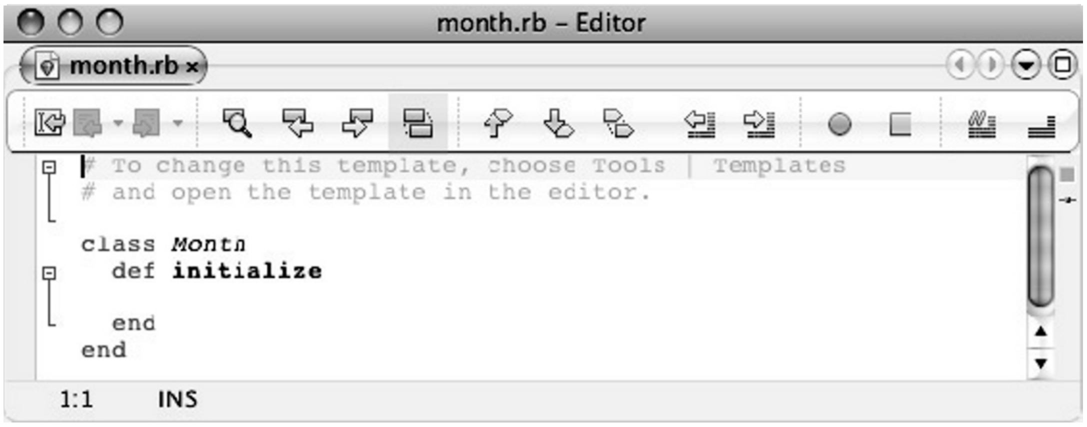


Class: Month
File Name: month
In Module:
Extends:
Project: calendar
Location: Source Files
Folder: /NetBeansProjects/calendar/lib
Created File: /NetBeansProjects/calendar/lib/month.rb

2. Give your class a name.
3. The IDE suggests a value for File Name but you can change it.
4. You can optionally specify whether your new class is in a module or extends a class.
5. For the Location value, the IDE suggests Source Files, Test Files, or RSpec Files, based on the type of file that you are creating. For class files, the IDE suggests Source Files, but you can choose one of the other options.
6. The IDE suggests a Folder based on the Location. If you want to put the class in a different folder, type the path to the new folder, or click Browse to navigate to and select a different location.

7. Click Finish. The IDE creates the file and opens it in the editor (see Figure 4-7). You will learn about the IDE's editing features in Chapter 6.

Figure 4-7. Newly Created Class in the Editor Window



```
# To change this template, choose Tools | Templates
# and open the template in the editor.

class Month
  def initialize

  end
end
```

The IDE also provides templates for YAML, Ruby Unit Test, Ruby Unit Test Suite, and RSpec files, plus several other more generic types of files.

If you want to create a different type of file than the choices offered in the New submenu, choose New ➤ Other to open the New File wizard. The wizard offers many nonRuby choices, such as HTML and XML documents. If you don't see the type that you want, choose Other in the Categories list, and choose Empty File in the File Types list.

Running Ruby Applications

Following are some ways to run your application from the IDE:

- Right-click the project's node and choose Run from the pop-up menu.
- Choose Run ➤ Run Project from the main menu to run the current project (this is the project that you last performed an action on). The menu item shows the name of the project that the IDE will run.

- Press F6 or click the Run button that is in the main toolbar to run the current project (as identified in the IDE's toolbar).
- If a Ruby file (`.rb`) is open in the editor, you can right-click in the source and choose Run from the pop-up menu to run that specific file.

Tip There is actually another way to run the project. Choose Actions from the Quick Search box in the main toolbar so that `Search in Actions` appears in the Quick Search text box. Type **ru** to see all the IDE actions that have ru in their names. Click Run Project to run the current project. This search box is a good way to quickly find the action that you want to perform.

You use the Run category in the Project Properties dialog box to specify how to run the project. To open the dialog box, right-click the project's node in the Projects window and choose Properties. Then select the Run category (see Figure 4-8), and specify the following settings:

Main Script. This is the class that the IDE executes when you run the project.

Arguments. Use this text box to pass runtime arguments to the main script through the `ARGV` array.

Working Directory. By default, the IDE runs the application in the Source Files directory. If you want the project to run in a different directory, specify the path here.

Ruby Options. This is where you pass command-line switches for the Ruby interpreter. To learn the available switches for your interpreter, enter `-h` in the text box and run the application. The Output window lists the switches. Some commonly used switches are as follows:

-c: Check syntax only

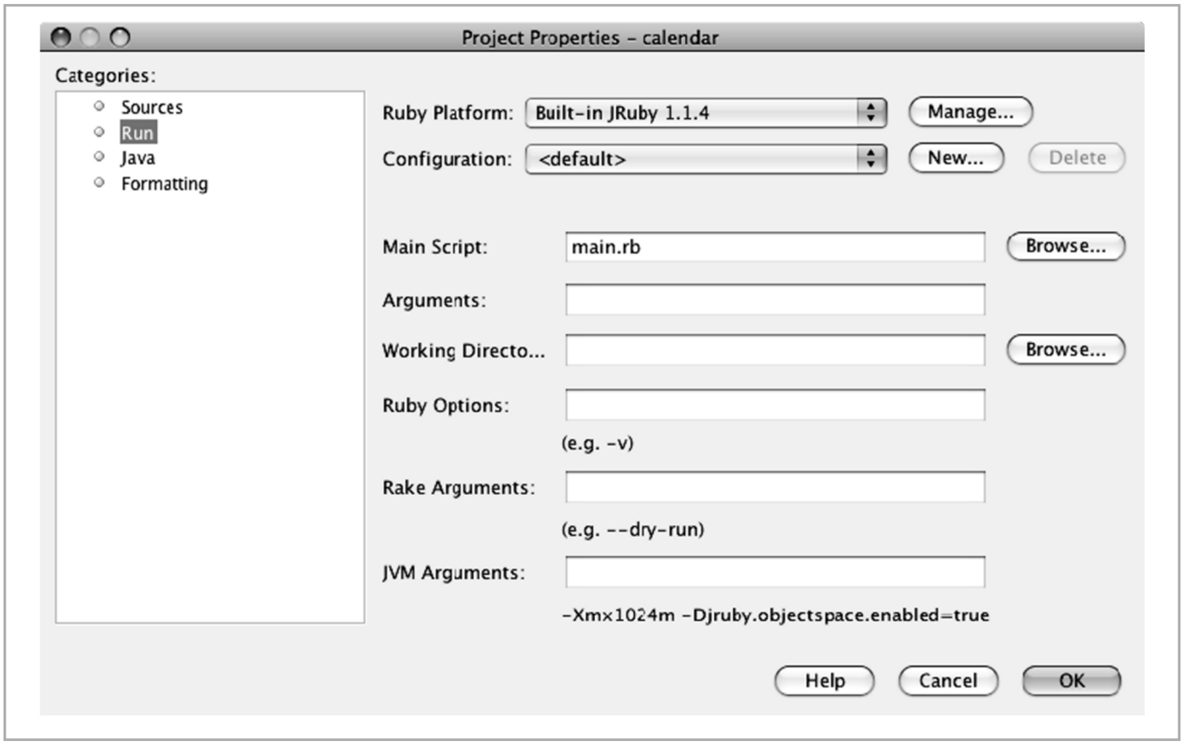
-w: Turn warnings on for your script

-d: Set debugging flags

JVM Arguments: For JRuby applications, use this text box to pass options to the JVM. To learn more about JRuby properties, see Chapter 8.

Note that the Rake Arguments setting does not apply to running projects. You use this setting for running Rake tasks, which you learn about next.

Figure 4-8. Run Category in Project Properties Dialog Box



Project Properties – calendar

Categories:
- Sources
- Run
- Java
- Formatting

Ruby Platform:	Built-in JRuby 1.1.4
Configuration:	<default>
Main Script:	main.rb
Arguments:	
Working Directo...	
Ruby Options:	(e.g. -v)
Rake Arguments:	(e.g. --dry-run)
JVM Arguments:	-Xmx1024m -Djruby.objectspace.enabled=true

Tip You can have multiple Run configurations. To add a configuration, click New, type a profile name, then click OK. The IDE adds the configuration to the Configuration drop-down list and makes that the *current* configuration. You can quickly change the *current* configuration from the Projects window by right-clicking the project’s node and choosing Set Configuration. Then select the desired configuration.

Running Rake Tasks

When you create a new Ruby project, the IDE includes basic Rake tasks such as `clean`, `clobber`, and `rdoc`. The following steps show how to run Rake tasks from the IDE.

1. To pass options to the Rake command, such as `--dry-run` or `--trace`, right-click the project's node and choose Properties from the pop-up menu. Select the Run catego ry for Ruby projects and select the Rails category for Rails project, then enter the options in the Rake Arguments text box.

Tip The project node's pop-up menu has shortcuts to many Rake tasks such as Build Gem, Clean and Build Gem, Clean, Generate RDoc, and Test. In addition, the main menu has a Build Gem button and a Clean and Build Gem button.

---

2. Right-click the Projects node and choose Run/Debug Rake Task from the pop-up menu. The dialog box opens with a list of the project's Rake tasks (see Figure 4-9).
3. To narrow the list, type a few characters in the Filter text box (you can use the `?` and `*` wildcards).
4. Use the Parameters text box to pass key/value pairs (for example **RELEASE=2.1**) to the Rake task. The IDE reuses the parameters that you specified the last time you ran the task, so remember to clear out these values if you don't want to pass any parameters.
5. If you are having problems using a Rake task, you can select the Debug check box and debug that task just as you would debug an application. See Chapter 7 for more information about using the IDE's debugging features.
6. To run the task, double-click it or select the task and click Run. The Output window displays the results. You can rerun the task by pressing Alt+Shift+R (Ctrl+Shift+R on the Mac) and then pressing Enter, or by clicking the Rerun button in the Output window.

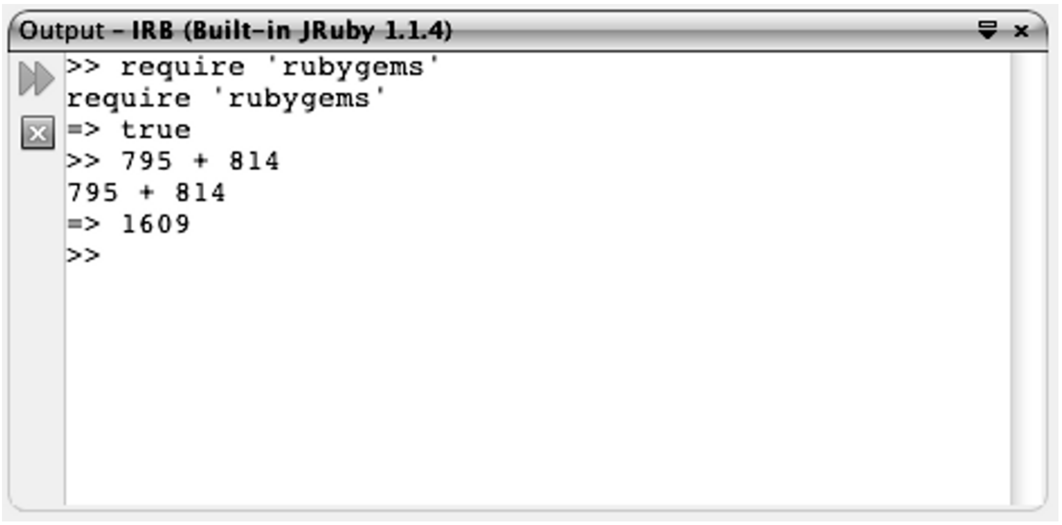
You can add your own tasks to the project's `Rakefile` and they will show up in the Run/Debug Rake Task dialog box. You might need to click Refresh Tasks to make your tasks appear.

Note By default, the dialog box only lists the tasks for which there is a description (for example `desc "Build all the packages"`). That is, it runs the `rake --tasks` command to obtain the list of Rake tasks. If you also want to see the tasks that do not have `desc` statements, select the Show Undocumented Tasks check box. The IDE will then run the `rake --describe` command instead.

Using Interactive Ruby (IRB)

The easiest way to test code from the IDE is to type the code into a class file and run that file by pressing Shift+F6.You can also open the Interactive Ruby (IRB) console from the IDE, which lets you interactively enter Ruby statements. You open the console by right-clicking the Ruby project's node and choosing Ruby Shell (IRB) from the pop-up menu. The IDE launches the console from the project's folder and displays it in the Output window, as shown in Figure 4-9.

Figure 4-9. IRB Console



```
>> require 'rubygems'
require 'rubygems'
=> true
>> 795 + 814
795 + 814
=> 1609
>>
```

The IDE looks at the Ruby Platform value in the project's Project Properties to determine which interpreter to use. The console tab shows which interpreter the shell is using.

In order to use gems in the IRB, you have to load Ruby Gems. While some Ruby interpreters do this for you, the JRuby interpreter does not. Here are a few ways to load Ruby Gems:

- Type **require 'rubygems'** in the IRB console.
- Set the Ruby Options text box in the Project Properties dialog box to **-rubygems**.
- Add an `.irbrc` file to the project's top-level folder and add `require 'rubygems'` to that file. Note that, by default, the IDE does not show files that begin with a dot (.). See Chapter 10 to learn how to configure the IDE to display dot files.

Summary

Creating Ruby projects is easy from the IDE. The New Project wizard guides you through the necessary steps, and, for projects that you are building from scratch, the IDE creates the basic directory structure and files that you need for a runnable application. You use the New File actions to add more files to your project, and the IDE provides many Ruby file templates to choose from. Being an integrated development environment, you do not need to leave the IDE to run your project or run Rake tasks on your project. You can even run the IRB console from the IDE.

Chapter 5: Working with Rails Projects

Ruby on Rails is an open-source framework for developing web applications that are built around data sources such as database tables. You work with Rails projects in the IDE similar to the way you work with Ruby projects (all project types in the IDE follow similar patterns). However, for Rails projects, the IDE provides additional features that integrate with the underlying Rails framework, as you will learn in this chapter.

NETBEANS IDE VS. THE COMMAND LINE

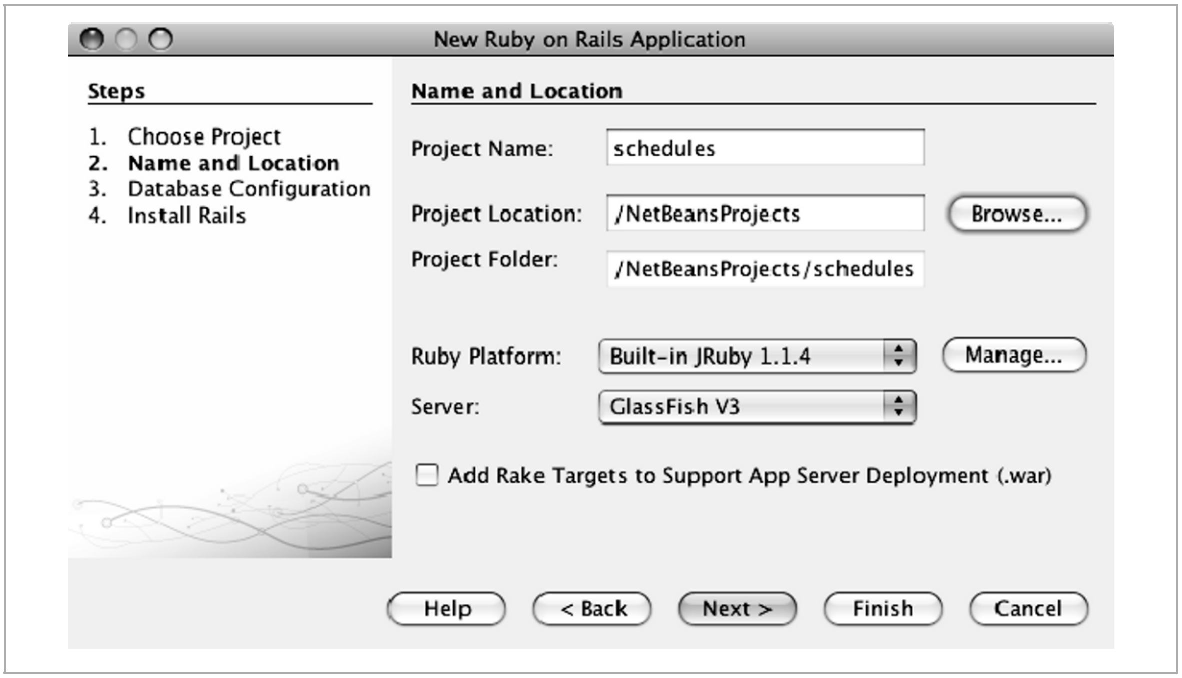
It's important to remember while working with the NetBeans IDE and Ruby on Rails that the NetBeans IDE is simply a veneer (albeit a very powerful one) on top of the existing Ruby on Rails framework. Most of the actions that are run by the IDE (creating a new project, running a generator, and so on) are calling out directly to the underlying Rails tasks and scripts. So the wizards and dialog boxes that you see in the IDE are nice graphical interfaces to the parameters and options you'd alternatively have to specify on the command line. We often find ourselves thinking, "Gee, I wish the IDE did such and such," only to realize it's really the Ruby on Rails script that we wish had that feature.

Setting Up Rails Projects

Similar to Ruby projects, the IDE enables you to create Rails projects from scratch as well as from existing sources. Here's how:

1. Select File ➤ New Project to open the New Project wizard.
2. Select Ruby in the Categories list.
3. In the Projects list, select either Ruby on Rails Application or Ruby on Rails Application with Existing Sources.
4. Click Next to go to the Name and Location step (see Figures 5-1 and 5-2).

Figure 5-1. Name and Location Step for Creating a Rails Application from Scratch



Project Name: schedules

Project Location: /NetBeansProjects

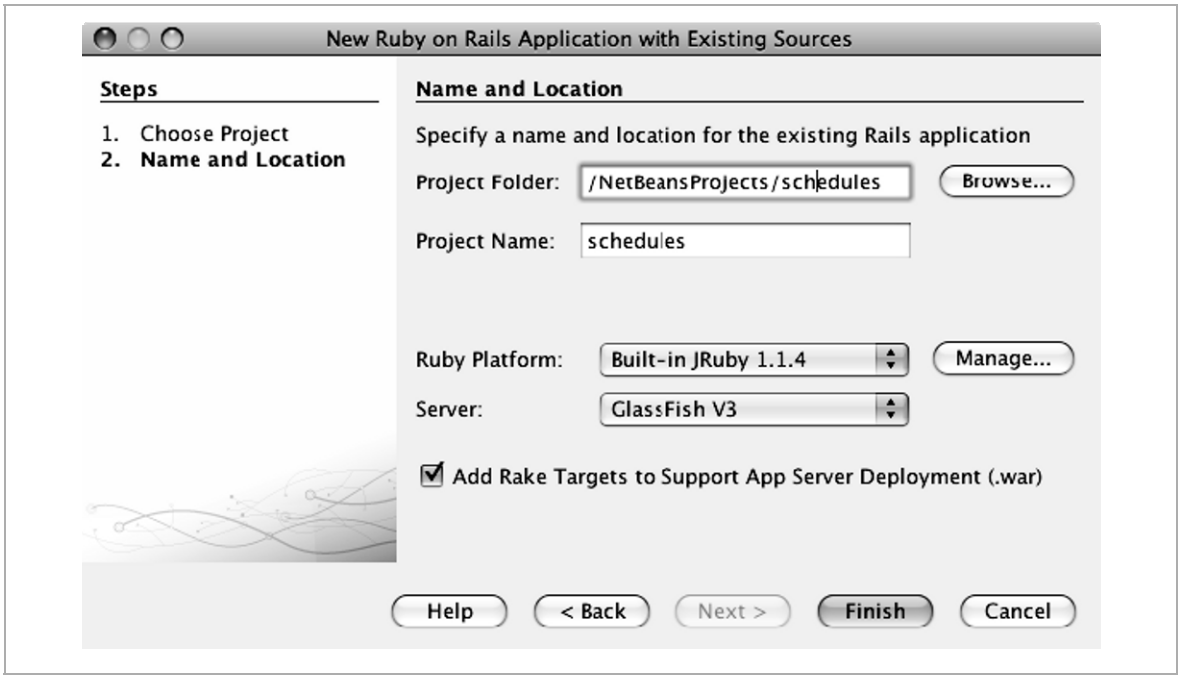
Project Folder: /NetBeansProjects/schedules

Ruby Platform: Built-in JRuby 1.1.4

Server: GlassFish V3

☐ Add Rake Targets to Support App Server Deployment (.war)

Figure 5-2. Name and Location Step for Creating a Rails Application from Existing Sources



Specify a name and location for the existing Rails application

Project Folder: /NetBeansProjects/schedules

Project Name: schedules

Ruby Platform: Built-in JRuby 1.1.4

Server: GlassFish V3

☑ Add Rake Targets to Support App Server Deployment (.war)

5. Enter the project's name and location. For Ruby on Rails applications with existing sources, enter the path to the project.
6. Select the Ruby Platform that you want to use for this project. You can change this setting at any time using the Project Properties dialog box. If the desired platform is not in the drop-down list, click Manage to register the platform with the IDE.
7. Select the server to run the application on. You can change this later using the Project Properties dialog box.
8. If you expect to package your application in a web archive (WAR) file, you can select the Add Rake Targets to Support App Server Deployment check box to include the Warbler plugin and to add `war` Rake tasks to the project. If you check this box and you don't have the Warbler gem, the IDE disables the Finish button. When you get to the final page you can click the Install Warbler button to install the gem. After you install the gem the IDE enables

the Finish button. See Chapter 9 for detailed information about creating WAR files and deploying to application servers.

9. If you are building a project from existing sources, click Finish. For projects that you are creating from scratch, you can either click Finish to go with the default configurations (which would be equivalent to running `rails` *`project-name`* `-d mysql` from the command line), or you can continue to the subsequent wizard steps to complete the following tasks:

   Specify database information. By default, the IDE edits the `database.yml` file to use the MySQL database server. It sets the development database name to *`application-name`*`_development` and sets the database names for test and production using the same pattern. If you are not using the default database configurations you can specify the configurations in the “Database Configuration” step of the wizard, or you can edit the `database.yml` file after the project is created. See the “Setting Database Configuration” section for details about using the “Database Configuration” step in the wizard.

   Specify the Rails version. If you have more than one Rails version in the interpreter’s gem repository, you can choose which Rails version to use in the “Install Rails” panel.

   Install or update the Rails gem. The “Install Rails” panel in the wizard displays the current Rails version number. You can click the Install/Update Rails button to install the latest version.

   Install or update the JRuby OpenSSL gem. If you're building a JRuby Rails application that requires a secure sockets layer (SSL), that is, an "https:" secure login, you'll need to have encryption libraries available. The JRuby OpenSSL support is not included in the bundled JRuby. You can click the Install/Update JRuby OpenSSL Support button in the “Install Rails” step to install the latest version.

   Install or update the Warbler gem. You can click the Install/Update Warbler button in the “Install Rails” step to install the latest version of the Warbler gem, which you can use to assemble a Rails application into a Java Web Archive (`.war`) file.

Working with Databases

How you work with databases in Rails applications is largely dependent on which interpreter you are running. With the CRuby interpreters, you are pretty much limited to the list of servers for which there are adapters (see `http://wiki.rubyonrails.org/rails/pages/DatabaseDrivers`). However, if you're running on JRuby, you now have access to JDBC, for which drivers exist for just about every database imaginable. Throughout the rest of this section we discuss the various options for configuring your database.

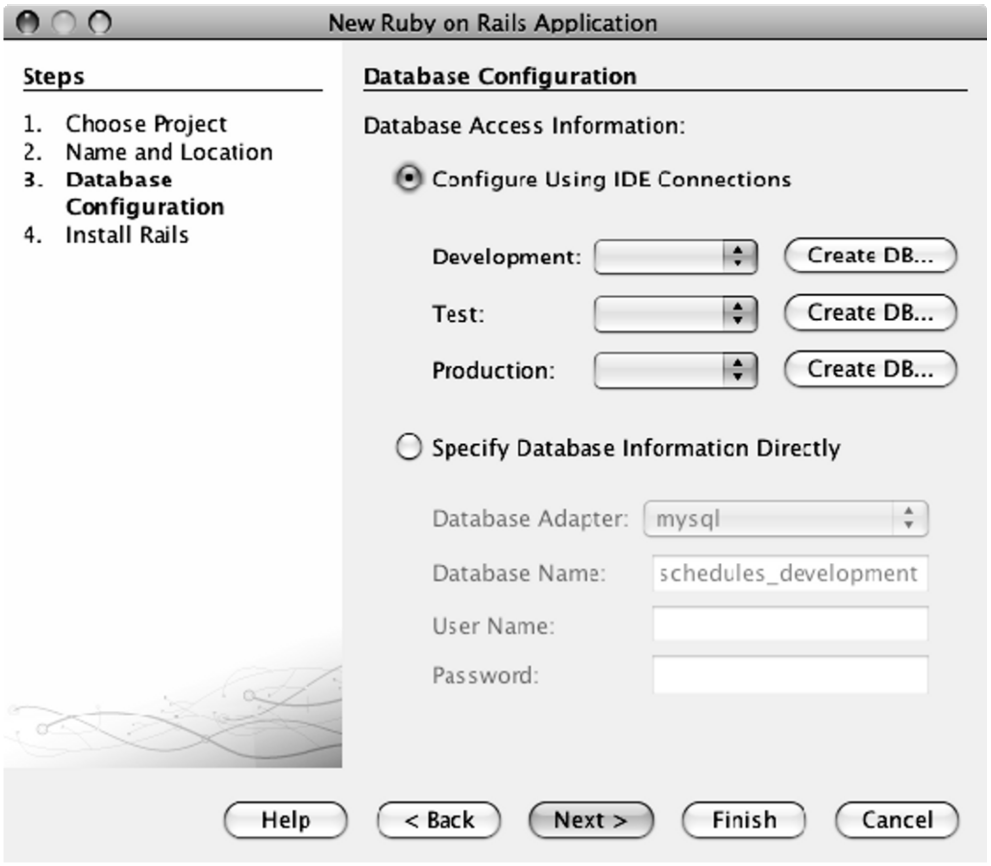
Note When you create Rails projects from the IDE, the database adapter is set to MySQL by default. Most Ruby platforms, including the JRuby platform, work out of the box with the MySQL adapter setting. For other adapter settings, you must install the appropriate gem. If you are using the gem in a JRuby application, the gem must not use native extensions. For example, for the PostgreSQL database you can use the Postgres PR gem with JRuby applications, because it is pure Ruby. If you cannot find a gem for your database server that does not use native extensions, you can use the generic Active Record JDBC Adapter or you can use a database-specific JDBC adapter as explained later in this section.

Setting Database Configurations

When you create a Rails project, the IDE modifies database configurations in the `database.yml` file based on your choices and selections in the New Project wizard (see Figure 5-3). When you create a new project you can specify the configuration settings in one of these ways:

- Use the default configurations
- Configure the database connections using IDE connections
- Specify the database information manually

Figure 5-3. Database Configuration Step in the New Project Wizard



Database Access Information:

[✓ Configure Using IDE Connections]

Development: Create DB...

Test: Create DB...

Production: Create DB...

Specify Database Information Directly

Database Adapter: mysql

Database Name: schedules_development

User Name:

Password:

The following sections explain each of these options.

Using Default Database Configurations

If you click Finish without accessing the "Database Configuration" step of the wizard, or without making any changes in that step, the IDE assumes that you are using the MySQL database server and you are following Rails conventions for your database names. That is, you have three separate databases for development, testing, and production purposes, and that the database names are composed of the project name plus the suffix `_development`, `_test`, and `_production`. Listing 5-1 shows an example of a default development configuration.

Listing 5-1. Default Development Database Configuration

```
development:
  adapter: mysql
  encoding: utf8
  database: cookbook_development
  username: root
  password:
  host: localhost
```

Note If you don't want to use root as the username, or if root requires a password, you need to edit the `database.yml` file after the project is created.

If you are using a different database server or different database names, you can edit the `database.yml` file after you create the project, or you can use the "Database Configuration" step to supply the information, as described next.

Configuring Database Connections Using IDE Connections

If you continue past the "Name and Location" step of the New Ruby on Rails Application wizard, you are presented with a couple of options for configuring the database connections. The first option is to use the connection information from a database that is already registered with the IDE (you learned how to register databases with the IDE in Chapter 3). To use this option, choose Configure Using IDE Connections in the "Database

Configuration" step (see Figure 5-3), and then select the registered databases from the drop-down lists. The IDE uses the port, username, and password that you used when you registered the database.

Tip If you are using a MySQL database, you can create and register a development, test, or production database directly from the "Database Configuration" step by clicking the appropriate Create DB button.

Specifying Database Information Manually

You can use the Specify Database Information Directly option in the "Database Configuration" step to provide information for the `database.yml` file. Select the Database Adapter, and enter the Database Name, User Name, and Password values for the development database. When you use this option, the IDE sets the adapter (and sometimes host or port depending on the adapter), username, and password for all three configurations. The IDE derives the database names for the test and production configurations from the development database name.

Note SQLite is not a client-server management system. Instead it is an embedded engine that writes the entire database in a flat file. Therefore, if you are using a SQLite database and you are running the application on the GlassFish application server, you must specify the full path to the database, such as this: `database:/sqlite/var/myapp_dev.sqlite3`.

Using Databases with JRuby

A MySQL database adapter is included with JRuby. For other database servers, as long as you have a database adapter gem that does not use native extensions, you can configure your `database.yml` file just as you would with CRuby applications. For example, you can use the postgres-pr adapter because it is pure Ruby. As of the time of this writing, most other database adapter gems include native extensions. When you try to install the gem into the JRuby gem repository, you will get the message `ERROR: Failed to build gem native extension`. If the only available adapters for your database server use native extensions, you must set up your configuration using a JDBC option.

Note For JDBC, you'll find database-specific JDBC adapters specifically written for JRuby. We list a few in the next section. In the case where a database-specific JDBC adapter does not exist for your database, you can optionally use the generic Active Record JDBC Adapter, which we talk about later in this section.

Using Database-Specific JDBC Adapters

There are several database-specific JDBC adapters available for JRuby, such as those in the following list. These adapters are convenience wrappers around the generic Active Record JDBC Adapter, which we talk about next. The database-specific adapters include the necessary drivers, and they enable you to specify the database configuration without having to know the details, such as the JDBC URL and driver class name.

Java DB/Apache Derby. activerecord-jdbcderby-adapter

H2. activerecord-jdbch2-adapter

HSQLDB. activerecord-jdbchsqldb-adapter

MySQL. activerecord-jdbcmysql-adapter

PostgreSQL. activerecord-jdbcpostgresql-adapter

SQLite. activerecord-jdbcsqlite3-adapter

Use the middle part of the gem name for the configuration's adapter setting. For example, you use `jdbcpostgresql` as the adapter for PostgreSQL.

Using the Generic Active Record JDBC Adapter

As mentioned earlier, you can use the Active Record JDBC Adapter gem (this is already installed with the bundled JRuby) to establish a JDBC connection with your database. To use this gem, you must put the server's JDBC 3.0–compliant driver JAR file in your classpath. You can use the Java category in the Project Properties dialog box to add JAR files to a project's classpath, as described in Chapter 8. If you select the Access Database Using JDBC check box when you create the project, and you have registered that database server with the IDE, the IDE puts the appropriate driver in the classpath for you. See Chapter 3 for information about registering database servers with the IDE.

Note If you are running the application in the GlassFish V3 server, you might also need to copy the driver JAR files to the `lib` folder in your JRuby installation. If you get an error message such as `The driver encountered an error: cannot load Java class org.postgresql.Driver`, copy the driver's JAR file into the `lib` folder and restart the server. This is a known issue that will be addressed in a patch or future version.

If you are using the generic JDBC adapter, you can configure the `database.yml` file in one of two ways:

- You can set the adapter to `jdbc` and specify the driver and URL as shown in Listing 5-2.

Listing 5-2. Development Configuration Using JDBC

```
development:
  adapter: jdbc
  driver: com.mysql.jdbc.Driver
  url: jdbc:mysql://localhost:3306/cookbook_development
  encoding: utf8
  username: dbadmin
  password: secret
  host: localhost
```

- You can add the statements shown in Listing 5-3 to your `environment.rb` file and configure the `database.yml` file just as you would for CRuby. This option enables you to run your application with either CRuby or JRuby.

Listing 5-3. JDBC Statements for the environment.rb File

```
if defined? JRUBY_VERSION
   require 'rubygems'
   gem 'activerecord-jdbc-adapter'
   require 'jdbc_adapter'
end
```

Creating Databases

In Chapter 3, we showed you how to create MySQL and Java DB databases directly from the IDE. With Rails 2, you can also use the Rake `create:db` task to create databases using the configurations you specified in the `database.yml` file. To do this you simply right-click the project's node, choose Run/Debug Rake Task, and double-click either of the following tasks:

`db:create`. Creates the database for the configuration that is defined by `RAILS_ENV`. By default this is the development configuration. To create a database for a different configuration, set `RAILS_ENV` in the Parameters text box (for example, `RAILS_ENV=test`).

`db:create:all`. Creates all the local databases that are defined in the `database.yml` file.

If the task creates the databases successfully, you get no output. Sometimes, the task does not create the database but does not report any errors. One way to verify whether the task created the databases is to run the `db:schema:dump` task and look for the `schema.rb` file under the Database Migrations node.

Note Currently, the `db:create` tasks might not work when using an Active Record JDBC Adapter (that is, your adapter setting in the `database.yml` file is set to `jdbc` or `jdbc<database>`, such as `jdbcmysql`).

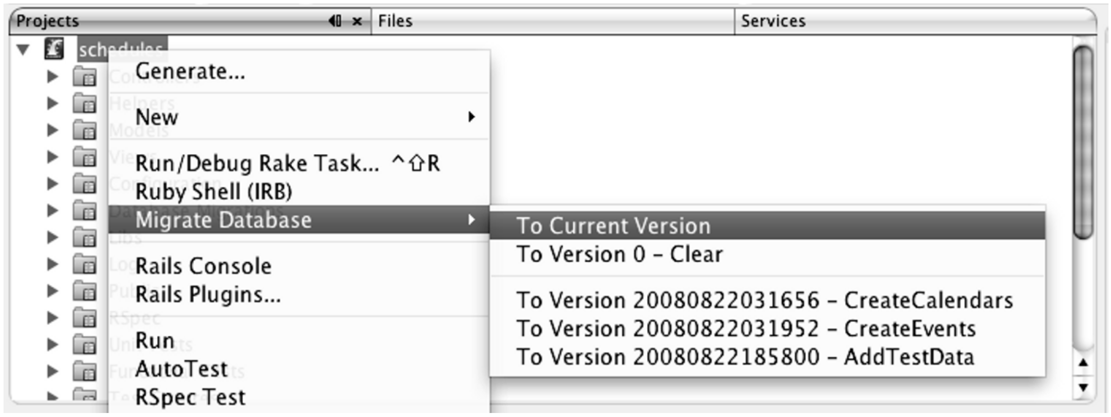
Creating and Changing Tables

As you learned in Chapter 3, you can create database tables using the Database Explorer. However, with Rails, migrations are generally used to achieve this task. Later in this chapter we discuss the Rails generator. Several of the commands that you execute from the generator, such as `generate model` and `generate scaffold`, create a model class and an initial migration class. When you execute the migration class it creates a database table. If you later want to make changes to the database table you create another migration class that, when executed, modifies the actual database table. These classes also provide code for backing out the changes (or deleting the table in the case of the initial migration).

There are several ways to run a migration from the IDE:

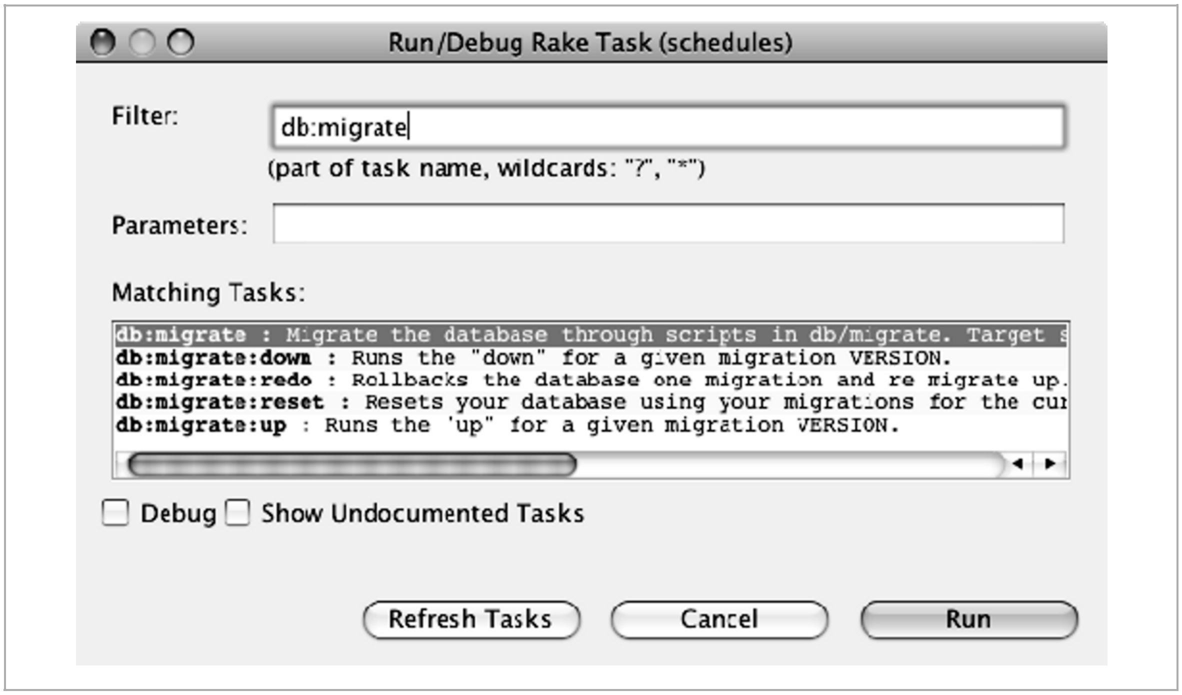
From the Project's Pop-Up Menu. Right-click the project's node and choose Migrate Database. As shown in Figure 5-4, the submenu contains an action for migrating to the current version and an action for migrating to version 0 (which backs out all the migrations). The submenu also lists all the migration files that exist under the Database Migrations ➤ Migrations Node. Choose the action that you want to migrate up or down to.

Figure 5-4. Migration Choices in the Pop-Up Menu



Using Rake. Right-click the project's node and choose Run/Debug Rake Task. Type **db:migrate** in the Filter text box to list the Rake migration tasks (see Figure 5-5).

Figure 5-5. Rake Migration Tasks


Run/Debug Rake Task (schedules)
Filter:
db:migrate
(part of task name, wildcards: "?", "*")
Parameters:
Matching Tasks:

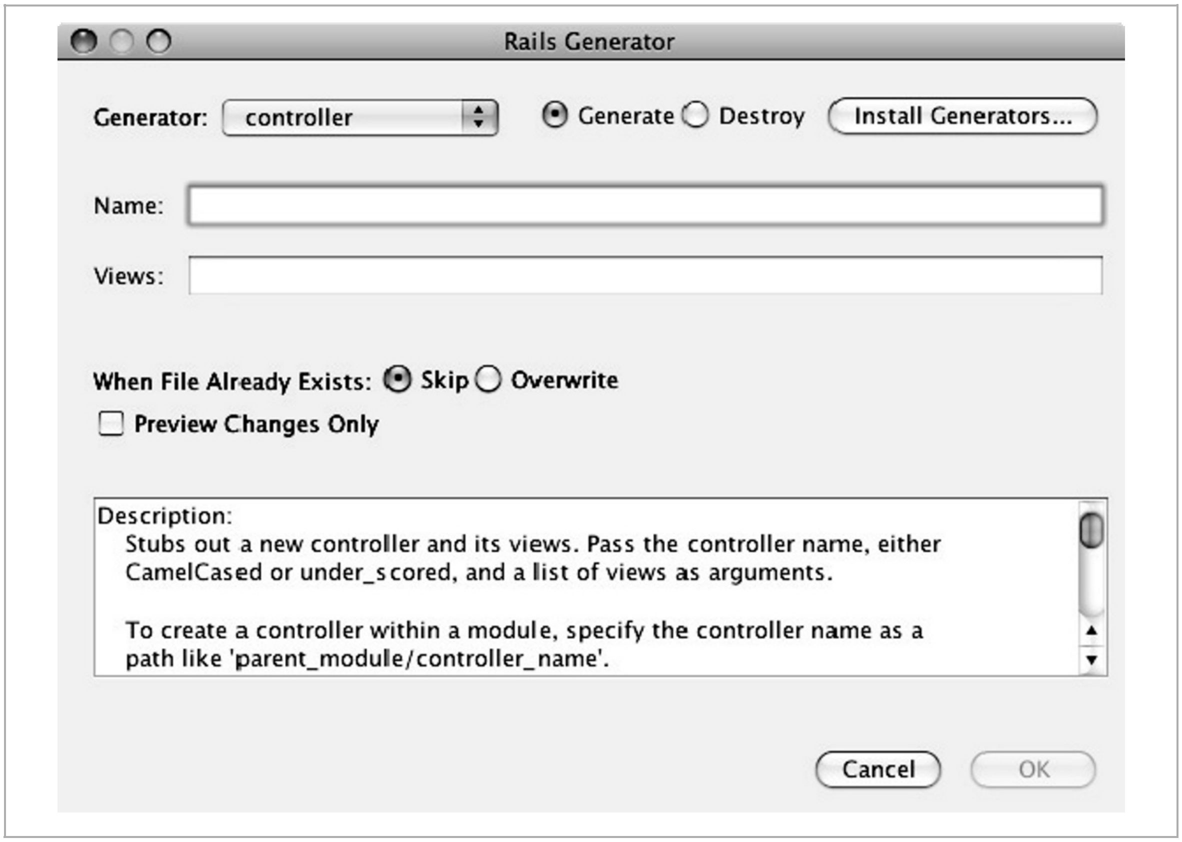
```
db:migrate : Migrate the database through scripts in db/migrate. Target s
db:migrate:down : Runs the "down" for a given migration VERSION.
db:migrate:redo : Rollbacks the database one migration and re migrate up.
db:migrate:reset : Resets your database using your migrations for the cur
db:migrate:up : Runs the 'up" for a given migration VERSION.
```

From a Migration File. If the class is open in the editor you can right-click in the class and choose Run File to run that migration.

Working with Generators

You use the Rails Generator to work with the Rails generate commands (see Figure 5-6). You can open the generator by right-clicking the project's node and choosing Generate.

Figure 5-6. Rails Generator Dialog Box



Generator: controller [✓ Generate] Destroy

Name:

Views:

When File Already Exists: [✓ Skip] Overwrite

☐ Preview Changes Only

Description:
Stubs out a new controller and its views. Pass the controller name, either CamelCased or under_scored, and a list of views as arguments.

To create a controller within a module, specify the controller name as a path like 'parent_module/controller_name'.

To use the Rails Generator, you first choose which generator to use from the drop-down list. The IDE then displays the settings you need to provide for that generator. Fill out the form, specify whether you want to overwrite existing files, and then click OK.

Tip You can add generators by clicking the Install Generators button. The Ruby Gems Manager opens and lists the available generators to install.

If you are not sure how to fill out a text box, look at the description panel, which shows the help text for that generator. You can also do a dry run of the command by selecting Preview Changes Only and clicking OK. The output window shows what the command would have done, but doesn't actually do it.

Notice that there is a Destroy radio button in the dialog box. As you would expect, you use this to back out the changes that were originally generated.

Tip You can also open the Rails Generator from the pop-up menu for the Controllers node, Models node, Views node, and Database Migrations node. When you open the Rails Generator from these nodes, the applicable generator is preselected. For example, if you right-click Database Migrations and choose Generate, the Rails Generator opens with the migration generator selected.

Adding Files

We pretty much covered what you need to know about adding files in Chapter 4. About the only difference with Rails is that you can use the Rails Generator to do a lot of the menial work for you.

Running Rake Tasks

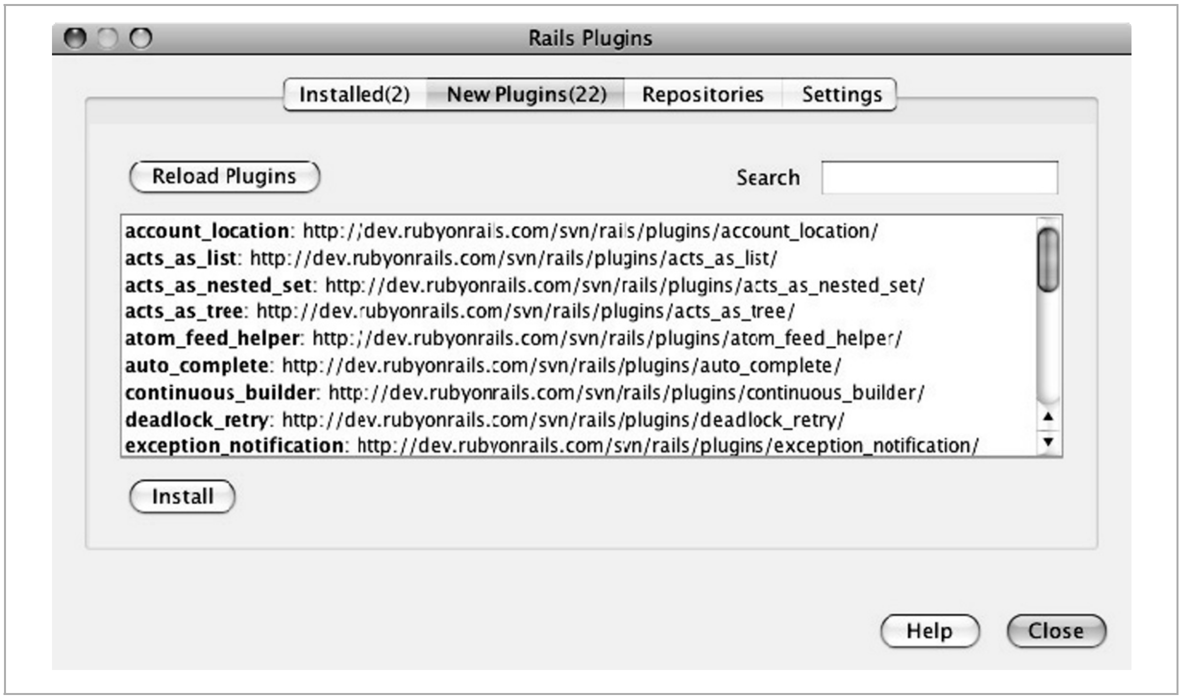
We covered the basics of how to use the Run/Debug Rake Task dialog box in Chapter 4. You will see a lot more tasks with Rails applications, such as the `db:create` task that you learned about earlier. The IDE makes some of these Rake tasks easier to get at by integrating them into the pop-up menus. The Migrate Databases action, which is a wrapper for the `db:migrate` task, is one example.

Adding Plugins

Similar to the Ruby Gems Manager, the IDE provides the Rails Plugins Manager for working with Rails plugins. To access the Rails Plugin Manager, right-click the project's node and select Rails Plugins.

Like the Ruby Gems Manager, it has a tab for viewing installed plugins and for installing new ones (see Figure 5-7). It also has a tab for specifying plugin repositories. If you don't see the plugin you are looking for, go to the Repositories tab and add the URL for the repository that contains that plugin.

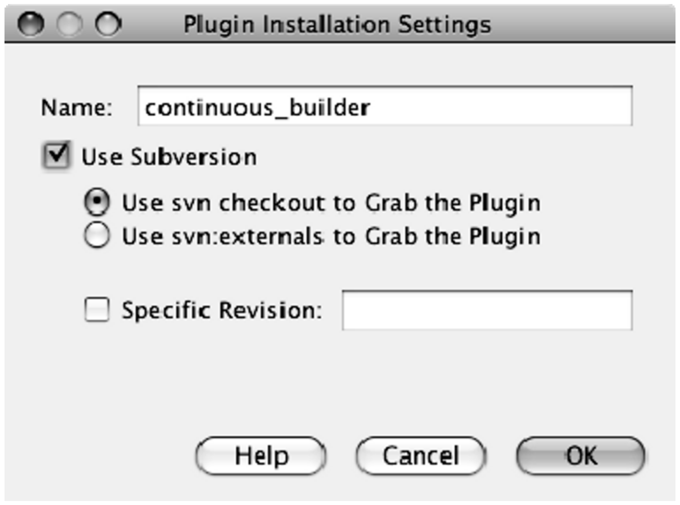
Figure 5-7. Rails Plugins Manager



account_location: http://dev.rubyonrails.com/svn/rails/plugins/account_location/
acts_as_list: http://dev.rubyonrails.com/svn/rails/plugins/acts_as_list/
acts_as_nested_set: http://dev.rubyonrails.com/svn/rails/plugins/acts_as_nested_set/
acts_as_tree: http://dev.rubyonrails.com/svn/rails/plugins/acts_as_tree/
atom_feed_helper: http://dev.rubyonrails.com/svn/rails/plugins/atom_feed_helper/
auto_complete: http://dev.rubyonrails.com/svn/rails/plugins/auto_complete/
continuous_builder: http://dev.rubyonrails.com/svn/rails/plugins/continuous_builder/
deadlock_retry: http://dev.rubyonrails.com/svn/rails/plugins/deadlock_retry/
exception_notification: http://dev.rubyonrails.com/svn/rails/plugins/exception_notification/

To install a plugin, select it in the New Plugins tab and click Install. If you want to use the Subversion version control to check out the plugin, select the Use Subversion check box and then specify whether to use svn checkout or svn:externals to get the plugin (see Figure 5-8). You can also specify a specific revision. When you click OK, the script installs the plugin in the project's `vendor/plugins` folder.

Figure 5-8. Plugin Installation Settings



Name: continuous_builder

[✓] Use Subversion

[● Use svn checkout to Grab the Plugin]

○ Use svn:externals to Grab the Plugin

☐ Specific Revision:

Running Applications

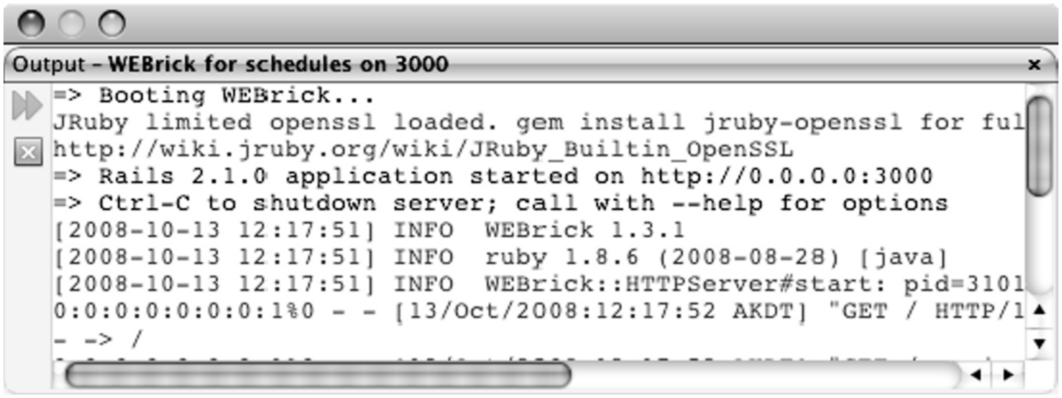
There are several methods for running a Rails application from the IDE:

- Choosing Run from the pop-up menu for a project's node
- Choosing Run ➤ Run Project from the main menu
- Pressing F6 or clicking the Run button in the main menu
- Pressing Shift+F6 from within a controller or view file. Note that within a controller this option is context sensitive. That is, the IDE attempts to run the action in which you placed the cursor. If you are using RESTful routes or the route requires data, the browser might display an error message such as `Couldn't find Recipe with ID=show`. When this happens you must rewrite the URL. For example, you would change

`http://localhost:3000/recipies/show` to
`http://localhost:3000/recipies/1`.

You usually do not need to rerun the application when you make changes to your application. You simply save the changes and refresh your browser. There are some circumstances where you need to restart the server, such as when you add JAR files to the classpath. To stop the server, click the red X on the left side of the server's Output window (see Figure 5-9). You can also stop the servers from the Services window. When you next run the application, the IDE restarts the server.

Figure 5-9. Server Output Window



```
=> Booting WEBrick...
JRuby limited openssl loaded. gem install jruby-openssl for ful
http://wiki.jruby.org/wiki/JRuby_Builtin_OpenSSL
=> Rails 2.1.0 application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2008-10-13 12:17:51] INFO  WEBrick 1.3.1
[2008-10-13 12:17:51] INFO  ruby 1.8.6 (2008-08-28) [java]
[2008-10-13 12:17:51] INFO  WEBrick::HTTPServer#start: pid=3101
0:0:0:0:0:0:0:1%0 - - [13/Oct/2008:12:17:52 AKDT] "GET / HTTP/1
- -> /
```

Tip See "Suppressing Browser Window Startup" in Chapter 10 to learn how to stop the IDE from opening up a browser window every time you run the project.

There are three Rails-specific project properties that affect how the IDE runs the application:

Rails Environment. By default, the applications run in development mode against the development database. If you are using WEBrick or Mongrel and you want to run in test mode, production mode, or any customer environment that you have set up in the `config/environments` folder, open the Project Properties dialog box and select the desired mode from the Rails Environment drop-down list. If you do not see the desired choice, you can just type it in. You must restart the server to make the change take affect. With NetBeans 6.5 this setting is not passed to the GlassFish V3 server. Instead, set `RAILS_ENV` in the `environment.rb` file, and then restart the server.

Server. When you create a project you choose which server to run on. You can change the server in the Project Properties dialog box. When you change the server setting you must stop all running servers in order for this setting to take affect. The IDE comes with the GlassFish application server, and WEBrick is bundled with Rails applications. If you install the Mongrel gem, it will also appear on the drop-down list. You can use the Server Port Number text box to specify the starting point for finding a free port for a Mongrel or WEBrick server. If that port is not available when the IDE starts the server, it increments the port by one until it finds a free port.

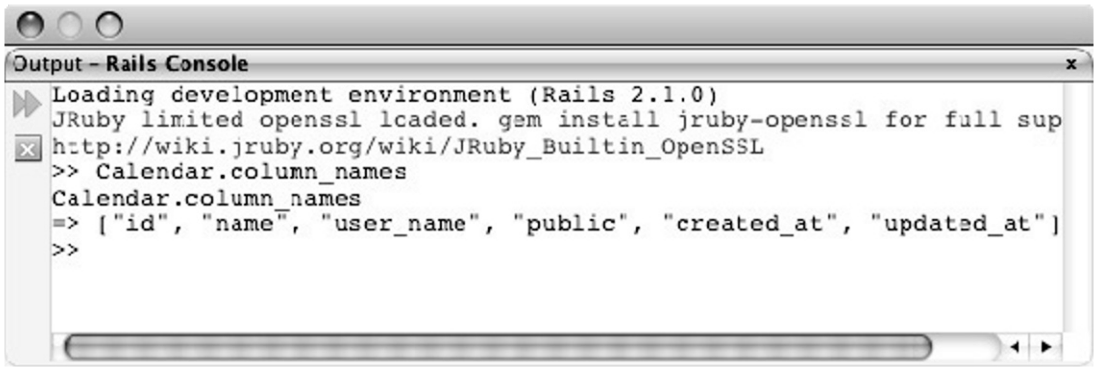
Encoding. The default encoding that is used by the IDE when writing files to disk is UTF-8. You can change this in the Project Properties dialog box.

Note If you have more than one JRuby installation on your system, you should be aware of how the GlassFish V3 server determines which JRuby installation to use. If you start the server from the Services window, the server uses the value in the Default JRuby Platform setting on the JRuby tab in the server's Project Properties dialog box. If you start the server by running a Rails application, the server uses that project's JRuby platform. Whichever platform the server is started with is the platform that the server uses for all the Rails applications until the server is restarted. If you have problems because of missing gems or libraries, it might be because you are not running with the project's JRuby interpreter.

Using the Rails Console

The IDE provides access to the Rails Console, which enables you to test out snippets of code. To open the console, right-click the project's node and choose Rails Console. A Rails Console tab appears in the Output window, as shown in Figure 5-10.

Figure 5-10. Rails Console



```
Loading development environment (Rails 2.1.0)
JRuby limited openssl loaded. gem install jruby-openssl for full sup
http://wiki.jruby.org/wiki/JRuby_Builtin_OpenSSL
>> Calendar.column_names
Calendar.column_names
=> ["id", "name", "user_name", "public", "created_at", "updated_at"]
>>
```

Summary

The NetBeans IDE has extensive support for working with Rails applications. When creating a new project, you can do so quickly by just providing a project name, or you can work through a series of wizard steps to customize your project, including which interpreter, web server, and database to use.

Configuring a database in Rails is typically a trivial operation. However, if you're using the JRuby interpreter, you now have the option to use JDBC, which opens up a new realm of possibilities. You'll find JDBC adapters that are specifically written for the more popular databases, such as MySQL and SQLite, as well as the option to use a generic JDBC adapter. Once your database is configured, the IDE can execute the Rake tasks for creating and running your migrations.

The IDE also provides a nice interface for working with the Rails generators, including the display of the standard Rails documentation on how best to use the generators.

Chapter 6: Editing Files

Most of a developer's time is spent editing code, and the IDE is full of features to make that time more productive. In this chapter, we dive into the editing features that help you write Ruby and Ruby on Rails applications.

Live Code Assistance

As you type, the source editor parses your code and works with you to minimize your efforts.

Delimiter Matching

The IDE automatically adds (and removes) ending delimiters for you, such as braces, parentheses, quote marks, `begin`/`end`, and `def`/`end` pairs. If you get lost in your nested code, place the caret on a delimiter and the IDE highlights the matching delimiter. If you place the caret on a delimiter and press Ctrl+[(Cmd+[on the Mac), the IDE jumps to the matching delimiter.

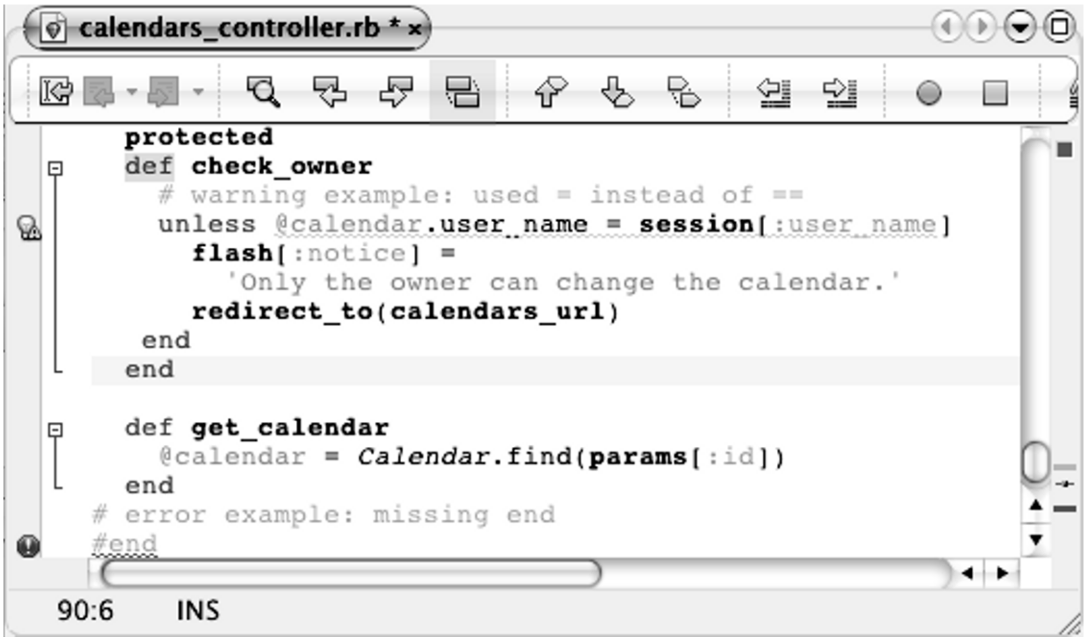
If you select some text in Ruby code and then type **(**, **{**, **[**, **‘**, or **“**, the IDE wraps the code with the beginning and ending delimiter. If you select some text in a Ruby comment and type **+**, **_**, or **“**, the IDE wraps the selected text with that character. To surround a string or regular expression with #{}, select the text inside of the “” or // delimiters and then type #.

Tip The IDE has a smart selection feature. If you press Alt+Shift+Period (Ctrl+Shift+Period on the Mac), the IDE selects the surrounding expression. For example if the caret is in a string, the IDE selects the string. Pressing Ctrl+Shift+Period again selects the next surrounding expression, such as the line, block, method, or group of comments, and so on, until it eventually selects the whole file. Use Alt+Shift+Comma (Ctrl+Shift+Comma on the Mac) to reverse (narrow) the selection.

Error Annotations

The IDE does background error parsing: when you stop typing, the IDE shows the errors along with descriptions of the errors. The IDE even models what an embedded Ruby (ERB) file will do in a Rails server and maps these errors back to the source. For example, it will report code errors in your scriptlets. The right-hand margin shows a red stripe for each line that has an error, and a yellow stripe for lines with warnings (see Figure 6-1). Clicking a stripe takes you to the line, and, when you hover over the error or warning icon in the left margin, the IDE shows a tooltip that describes the error. Later, you learn about the quick fix feature, which helps to correct some coding errors, and the Task List, which helps you track and access errors.

Figure 6-1. Error Annotations



```
  protected
  def check_owner
    # warning example: used = instead of ==
    unless @calendar.user_name = session[:user_name]
      flash[:notice] =
        'Only the owner can change the calendar.'
      redirect_to(calendars_url)
   end
  end

  def get_calendar
    @calendar = Calendar.find(params[:id])
  end
# error example: missing end
#end
```

With Ruby code, the IDE also identifies unused variables by underlining them with squiggly gray lines.

Formatting and Smart Indenting

The IDE indents and re-indents lines as you start and end methods and blocks. If your file's indentation gets misaligned, such as when you move around code, you can easily correct the indentation by right-clicking in the source and choosing Format from the pop-up menu. You can also use the Shift Line Left and Shift Line Right buttons in the editing toolbar to indent and unindent selected lines. When you paste a block of code, you can ask the IDE to automatically format the code you are pasting by using Ctrl+Shift+V (Cmd+Shift+V on the Mac).

The IDE also offers a feature for formatting comments. If your code comments are too long or are irregular in length, you can put the caret in the comment section and press Ctrl+Shift+P (Cmd+Shift+P on the Mac) to rewrap the comments to the line length that is designated by the right margin (the red line on the right shows the margin). The feature recognizes RDoc conventions. For example, it leaves preformatted code in RDoc comments alone and it recognizes and appropriately formats numbered lists.

Tip To customize how the IDE indents and formats your code, choose Tools ➤ Options, click Editor, click the Formatting tab, and choose Ruby from the Language drop-down list. You can also customize the formatting settings for a specific project. Right-click the project's node and choose Properties from the pop-up menu. In the Project Properties dialog box, select the Formatting category.

About Semantic Coloring

The source editor uses semantic coloring to make it easy to identify and locate the different parts of code. The following tables show what the different colors and fonts indicate in Ruby code and RDoc comments.

Table 6-1. Ruby Semantic Styles

ELEMENT	STYLE
Ruby keywords	Blue
Constants	Italic
Method calls	Bold
Parameters	Orange
Symbols	Cyan

Table 6-1. Ruby Semantic Styles (continued)

Element	**Style**
Regular expressions	Purple
Strings	Orange
Escapes such as \n and \C-x	Bold
Instance variables	Green
Class variables	Green italic
Unused variables	Underlined in gray
Embedded Ruby	Green background
Errors	Underlined in red

Table 6-2. RDoc Semantic Styles

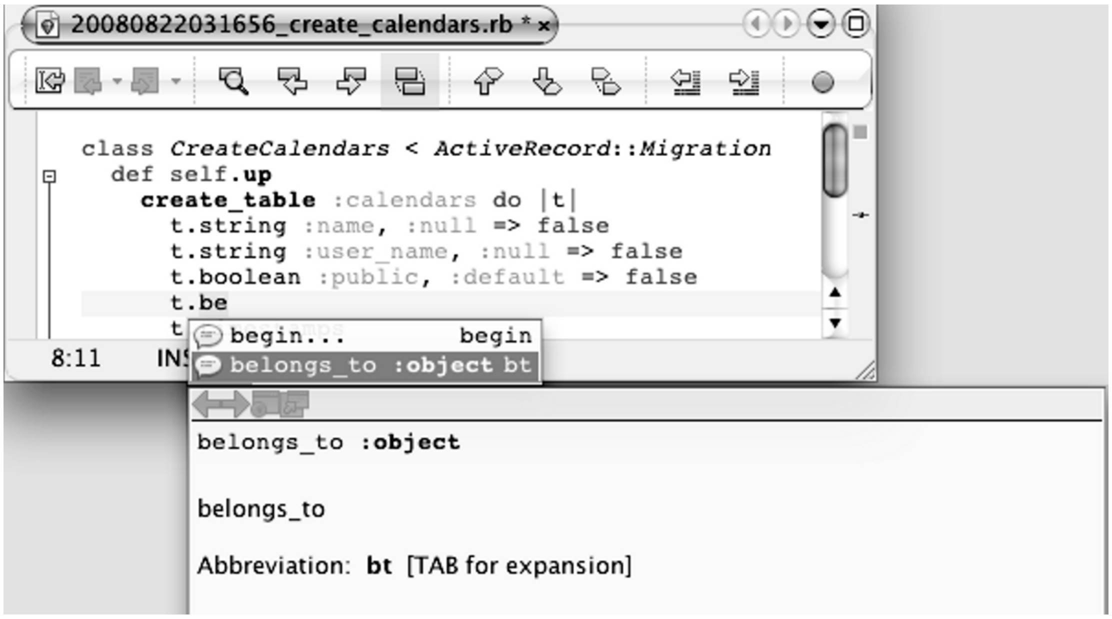
Element	**Style**
RDoc directive	Blue
Underline-surrounded word (_word_)	Italic
Asterisk-surrounded word (*word*)	Bold
Plus-surrounded word (+word+)	Monospace
References	Displayed as links

Tip The IDE offers several profiles, which have different semantic color schemes. Open the Fonts & Colors pane in the Options dialog box to switch to a different profile, look at the styles for all the elements, or customize the IDE with your own color schemes. See Chapter 10 to learn more.

Using Code Completion

The editor provides a context-sensitive pop-up that shows a list of code completion suggestions (see Figure 6-2).

Figure 6-2. Code Completion Suggestions



```
class CreateCalendars < ActiveRecord::Migration
  def self.up
    create_table :calendars do |t|
      t.string :name, :null => false
      t.string :user_name, :null => false
      t.boolean :public, :default => false
      t.be
      t
```

```
begin...              begin
belongs_to :object bt
```

```
belongs_to :object

belongs_to

Abbreviation: bt [TAB for expansion]
```

Note Because Ruby is a dynamic language and types are not declared, it is a bit harder to come up with this list then with a statically typed language such as Java, so the IDE looks at the project's usage patterns to compile the list of suggestions.

The code completion pop-up appears when you pause after typing a class, variable, or method followed by a period or double colon. You can also display the pop-up by pressing either Cmd+Space or Cmd+\ on the Mac or Ctrl+Space or Ctrl+\ on other operating systems (to minimize repetition we will just refer to this key combination as Ctrl+Space from here on). You can select from the pop-up list or continue typing to narrow the possibilities. The pop-up also shows the documentation, when it is available.

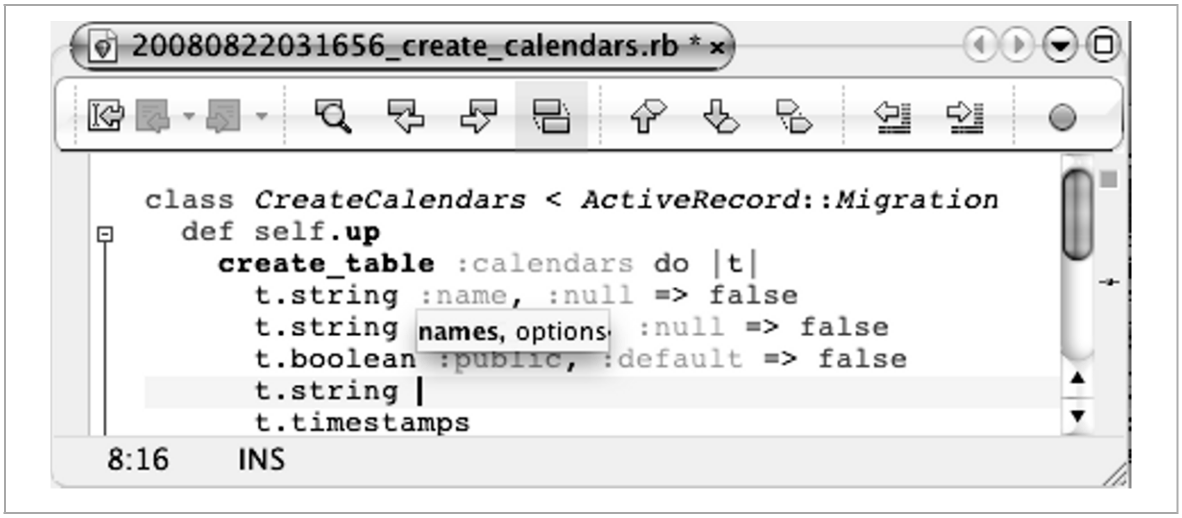
Tip If you just want to see the documentation, add Shift to the shortcut. That is, press Ctrl+Shift+Space (Cmd+Shift+\ or Cmd+Shift+Space on the Mac). See "Viewing RDoc" later in this chapter for more details.

Code completion works for literals just as it does for variables. For example you can type **[1,2,3].ea** and press Ctrl+Space to see the suggested array methods that begin with `ea`.

When you select a method from the code completion list, you can also fill in the arguments. After you fill in an argument, press Tab or Enter to go to the next argument, and then press Enter when you are done. If you are not sure what to enter for an argument, press Ctrl+Space for suggestions. You can even ask for suggestions for option arguments. For example, with the `add_column` method, you can type **:null** => and then press Ctrl+Space to see that the valid values are `true` and `false`.

If you are editing existing code you can also get parameter hints for a method (see Figure 6-3). Place the caret either inside the parentheses, just after the method name if you are not using parentheses, or on a parameter, and press Ctrl+P (Cmd+P on the Mac) to see the parameters. The current parameter being edited is shown in bold. Because it is not always possible to determine the method's signature, sometimes the tooltip will not appear.

Figure 6-3. Parameter Hints



```
class CreateCalendars < ActiveRecord::Migration
  def self.up
    create_table :calendars do |t|
      t.string :name, :null => false
      t.string       :null => false
      t.boolean :public, :default => false
      t.string |
      t.timestamps
```

Tip If you don't want the code completion pop-up to display automatically, you can turn it off using the Options manager. Choose Tools ➤ Options from the main menu, click Editor, click the General tab, and then clear the Auto Popup Completion Window check box. You can then display the pop-up on an as-needed basis by pressing Ctrl+Space (Cmd+Space on the Mac).

In addition to suggesting module names, class names, method names, and constants, the code completion will, depending on the context, make suggestions for the following:

- Local and dynamic variables in the current block
- Ruby built-ins such as keywords and predefined constants
- Global ($) variables
- Regular expression escape sequences
- Literal String (%Q) escape sequences
- Available imports
- Classes to inherit from
- Methods to override
- Escape codes following a percent sign (%)
- Database parameter completion in Active Record calls
- YAML special characters

Tip The IDE looks at a project's migration files and schema to compile its list of suggested model names and columns. If your project does not use migration files, you should run the `db:schema:dump` Rake task to create a `schema.rb` file.

Using Live Code Templates

Also known as snippets or abbreviations, live code templates enable you to type a few characters and then press Tab to expand the characters into a snippet of code. Some snippets contain logical parameters. Replace a parameter with the desired value and then press Tab or Enter to move to the next parameter.

Tip The far right column in the code completion pop-up shows the available triggers. For example, if you type **assert_** and press Ctrl+Space, the pop-up shows that `ase` is the trigger for `assert_equal`.

Table 6-3 shows some examples of code templates for Ruby. You can see the full list at `http://ruby.netbeans.org/codetemplates-ruby.html`.

Table 6-3. Examples of Code Templates for Ruby

DESCRIPTION	TRIGGER	SAMPLE EXPANSION
=>	l	`=>`
Hash Pair :key => "value"	:	`:parameter1 => "parameter2"`
assert(...)	ase	`assert_equal(parameter1, parameter2)`
belongs_to	bt	`belongs_to :parameter1`
collect { |e| ...}	col	`collect { |parameter1| }`
each { |e| ...}	ea	`each { |parameter1| }`
flash [...]	flash	`flash[$parameter1] = "parameter2"`

Table 6-4 shows some examples of code templates for ERB and RHTML. The complete list can be found at `http://ruby.netbeans.org/codetemplates-rhtml.html`.

Table 6-4. Examples of Code Templates for ERB and RHTML

DESCRIPTION	TRIGGER	SAMPLE EXPANSION
<% rubycode %>	R	`<%  %>`
<%= expression %>	Re	`<%=  %>`
HTML table	table	`<table border=”1><tr><td></td></tr></table>`

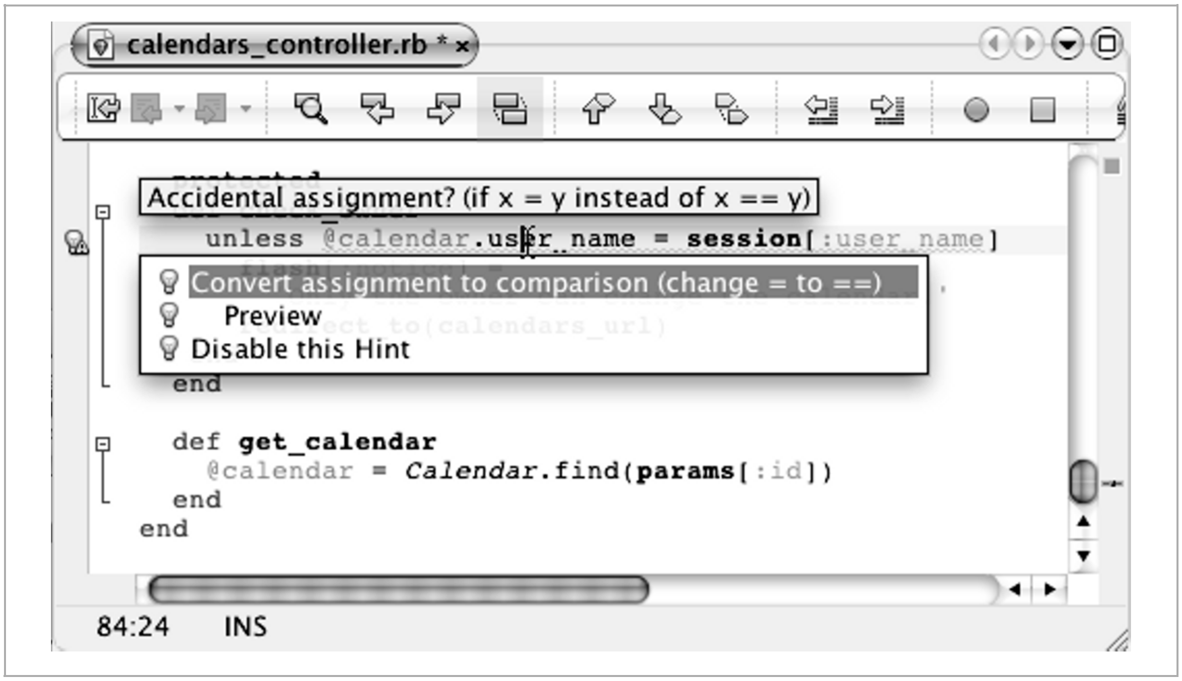
Some of the code templates, such as the Begin template, wrap selected code. In these cases you use the Hints mechanism to do the code expansion. You learn about Hints next.

Tip Chapter 10 will show you how to add and edit templates.

About Hints and Quick Fixes

The editor checks for common programming mistakes and code deprecations and provides hints and fixes to correct these errors. When you see an error or warning annotation for some code, you often see a small light bulb icon in the left margin. When you hover over the icon, a tooltip explaining the problem pops up. You can put the caret on the line and press Alt+Enter to see the suggested fixes (see Figure 6-4). Select the fix that you want and press Enter. If you are not sure you can select Preview to open a dialog box that shows what the fix will do.

Figure 6-4. Quick Fix Suggestions


calendars_controller.rb *
Accidental assignment? (if x = y instead of x == y)
Convert assignment to comparison (change = to ==)
Preview
Disable this Hint
84:24
INS

```
    unless @calendar.user_name = session[:user_name]
  end

  def get_calendar
    @calendar = Calendar.find(params[:id])
  end
end
```

As mentioned earlier, the editor also offers to wrap selected code with common code expansions. Whenever you select code, the light bulb icon appears. Pressing Alt+Enter displays several `Surround with` options such as `Surround with do {|e|}`, `Surround with =begin/=end`, and `Surround with if … | else … | end`.

The editor offers line-specific quick fixes for the more frequently occurring hints, such as offers to split single-line method definitions into multiple lines, or convert from a do/end block to a {} block. Because it would be messy to make these suggestions all over the file, these suggestions only appear when the caret is on the line.

To see the complete list of hints, choose Tools ➤ Options from the main menu, click Editor, click the Hints tab, and select Ruby from the Language drop-down list. You will notice that not all of the hints are enabled. You can select or clear the check boxes to customize which hints the editor displays. You can also specify which hints should display as warnings, which should display as errors, and which should not be displayed unless the caret is on the specific line.

As NetBeans engineers create new hints and quick fixes they add them to the Ruby Extra Hints plugin. These hints get incorporated into the next release. You might want to periodically install the Ruby Extra Hints plugin to pick up any new hints that have been added.

Tip You can use the Task List, which we talk about later, to apply quick fixes.

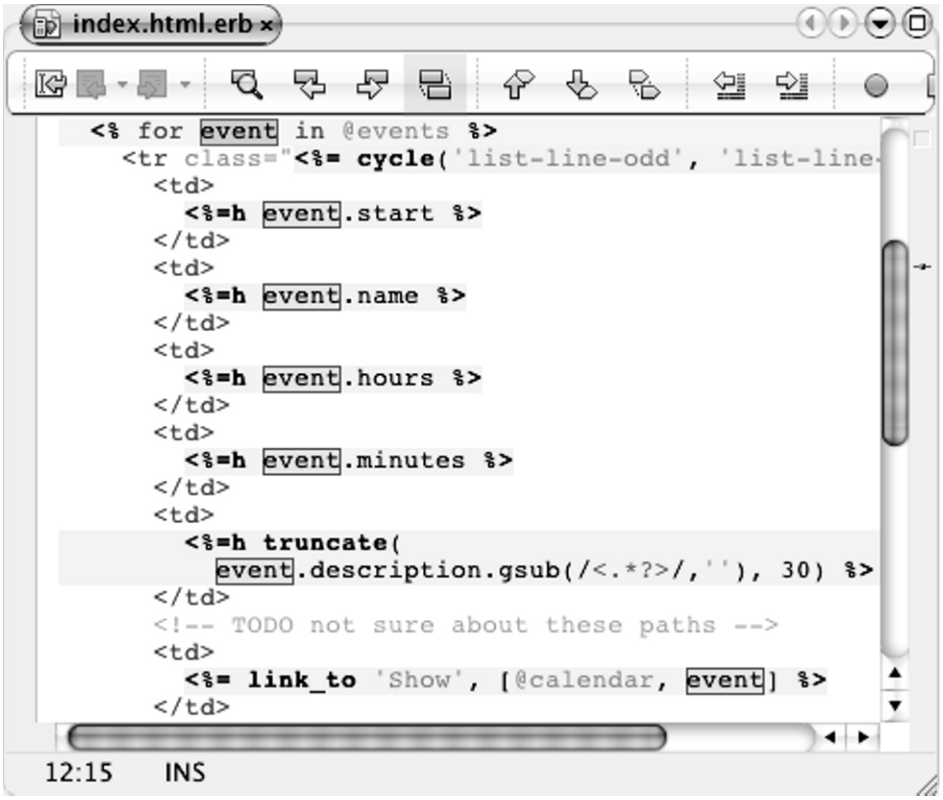
Refactoring Code

The IDE supports code refactoring in several ways. It enables you to easily rename elements, search and replace text, extract code into methods, and turn strings into constants or variables.

Renaming Elements

The refactoring feature you will probably find the most useful is inline renaming. You simply place the cursor on a local variable, press Ctrl+R, and then type the new name. As you type you can see the editor replace all usages of that variable with the new name (see Figure 6-5). Press Enter when you are done.

Figure 6-5. Inline Renaming

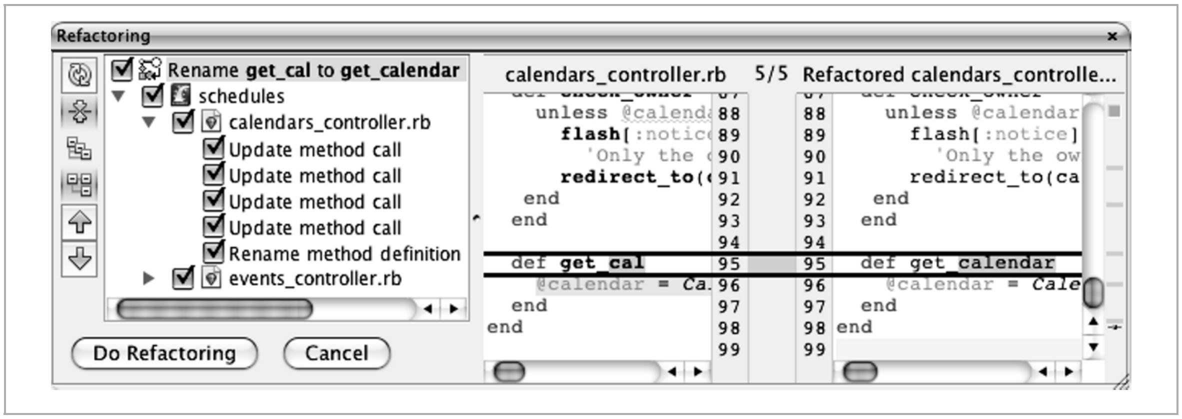


```
<% for event in @events %>
  <tr class="<%= cycle('list-line-odd', 'list-line-
    <td>
      <%=h event.start %>
    </td>
    <td>
      <%=h event.name %>
    </td>
    <td>
      <%=h event.hours %>
    </td>
    <td>
      <%=h event.minutes %>
    </td>
    <td>
      <%=h truncate(
        event.description.gsub(/<.*?>/,''), 30) %>
    </td>
    <!-- TODO not sure about these paths -->
    <td>
      <%= link_to 'Show', [@calendar, event] %>
    </td>
```

Renaming global variables, instance variables, class variables, and methods is almost as easy. When you press Ctrl+R on these elements, the IDE displays a dialog box. After you enter the new name, click Preview to see a list of places where the change will be made (see Figure 6-6). Click the Next Occurrence button in the left margin to review the before and after code for each line in the list. You can clear the check boxes of the lines that you do not want changed. When you are ready, click Do Refactoring.

Note Although the IDE will not confuse local variable references with method names, it is difficult to detect whether symbols and method names that occur in multiple places refer to the same method, so the IDE errs on the side of optimism and presents them all as potential uses.

Figure 6-6. Refactoring Preview Window



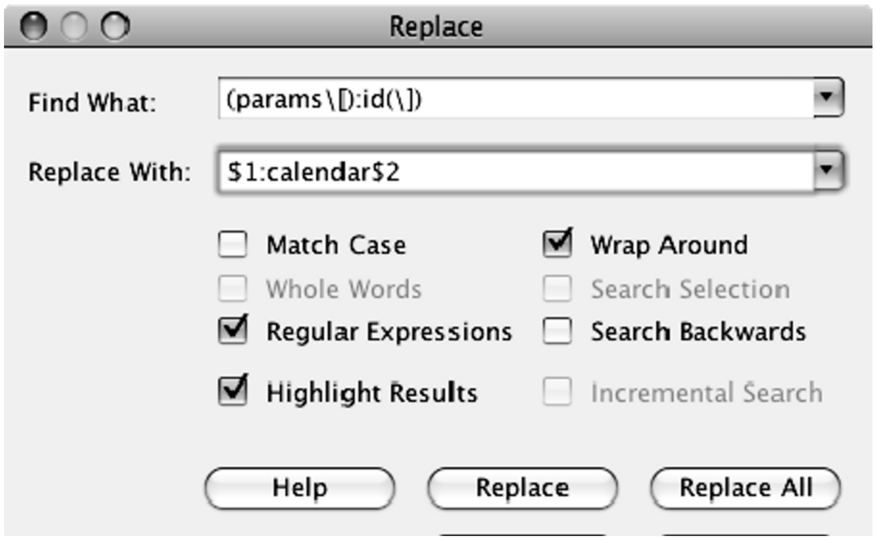
```
  unless @calenda
    flash[:notic
      'Only the c
    redirect_to(c
  end
end

def get_cal
  @calendar = Ca.
end
end
```

```
  unless @calendar
    flash[:notice]
      'Only the ow
    redirect_to(ca
  end
end

def get_calendar
  @calendar = Cale
end
end
```

Searching and Replacing

The IDE also has the classic search and replace dialog box, which you open by choosing Edit ➤ Replace from the main menu (see Figure 6-7).

Figure 6-7. Search and Replace Dialog Box



Replace

Find What: (params\[):id(\])

Replace With: $1:calendar$2

[ ] Match Case
[✓] Wrap Around
[ ] Whole Words
[ ] Search Selection
[✓] Regular Expressions
[ ] Search Backwards
[✓] Highlight Results
[ ] Incremental Search

One advantage of this dialog box is that you can use Java regular expressions to search for patterns. Following are some examples:

Newline. `\n`

Alphanumeric character. `\w`

Beginning of line. `^`

End of line. `$`

One or more. `+`

Zero or more. `*`

Delimiters for a set of characters. `[]`

Escape character for matching special characters. \

For the complete list click the Help button in the dialog box.

You can use back references in the Replace With text box to change parts of the found text. For example you can type **(params\[):id(\])** in the Find

What box and type **$1:calendar$2** in the Replace With box. The editor finds all occurrences of `params[:id]` and replaces the text with the first parentheses grouping `(params\[)`, plus the replacement text `:calendar`, plus the second grouping `(\])`. That is, `params[:calendar]`.

Extracting Methods

Another nice refactoring feature is the ability to extract code into a method. You simply select the code that you want to extract, press Alt+Enter, and select Extract Method. The Extract Method dialog box appears. Type the name of the method and click OK. A dialog box appears asking for the name of the new method. The IDE creates a new method with the selected code and places the cursor in the method's comment. The IDE figures out which variables are needed by the method and adds method parameters to pass the values in. The IDE also figures out what variables to pass back.

Tip Before selecting the Extract Method suggestion, you might want to first choose Preview to make sure you like the change.

Introducing a Constant, Field, or Variable

How often do you have strings or numbers in your code that you decide to turn into a constant, local variable, or instance variable? All you have to do is select the number or string (including the quote marks) and press Alt+Enter. You can then choose Introduce Constant, Introduce Field (instance variable), or Introduce Variable. A dialog box appears asking for the name of the constant or variable. If the IDE sees that you use the selected value more than once, it displays a check box asking whether to replace all occurrences. When you introduce a constant, the IDE adds a comment for the constant and places the cursor on that line.

Navigating Code

One of the advantages of using the IDE for your projects is that you can quickly and easily access all of a project's files. Not only do you have the Projects and Files windows to view your files in a hierarchical manner, but you have numerous options for finding and jumping to just the code you are looking for.

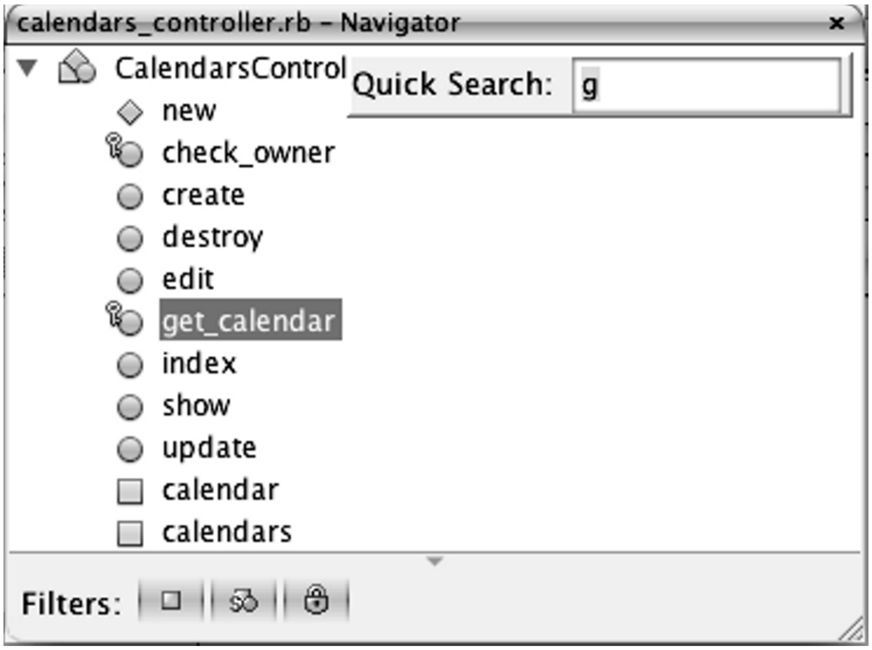
Navigating Within a File

Once a file fills up a screen, you might find yourself often scrolling up and down to look up a correct name, see where a variable is used, or find some code you want to copy. When moving about a file you will want to take advantage of the Navigator window and the bookmarking feature, as described here, as well as the navigation to member actions that are described in the next section.

Using the Navigator Window

When you open a file in the editor, the Navigator window (see Figure 6-8) lists the member modules, classes, fields, methods, and attributes. When you double-click on a node in this Navigator window, the editor scrolls to the code for that member. If you start typing in the Navigator window a Quick Search box appears. You can type the first few characters of a member name into the box until the desired member is selected in the Navigator window, then press Enter to navigate to the related source.

Figure 6-8. Navigator Window



Jumping to Bookmarks

If you find yourself jumping back and forth between certain lines, you can press Ctrl+Shift+M (Cmd+Shift+M on the Mac) on each line to bookmark that line (or remove an existing bookmark). The IDE shows a gray stripe in the right margin and a blue bookmark icon in the left margin for each bookmark. To jump to a bookmarked line, click its grey stripe in the right margin. You can also right-click in the left margin and choose Bookmark ➤ Next Bookmark, or choose Bookmark ➤ Previous Bookmark.

Navigating to Members

In addition to the Navigator window, the following features enable you to navigate to classes, fields, methods, and attributes within the file and within the project.

Navigate to Declaration

You can Ctrl+Click on a class, field, or method, to navigate to its declaration. If it is declared in a different file the IDE opens that file. You can also navigate to the declaration by right-clicking the class, field, or method and choosing Navigate ➤ Go to Declaration from the pop-up menu. In addition, you can use the Go to Type and Go to Symbol dialog boxes, which are described in the Finding Files section.

Note When you choose to navigate to the declaration and the IDE finds multiple possibilities, the IDE displays a list to choose from. It puts what it thinks is the best choice at the top of the list. The IDE uses factors such as which classes are directly loaded by your file using `require` statements, and what is documented to determine its best choice. The other entries are listed in alphabetical order, with the `:nodoc` entries at the end (shown with strikethroughs). If you want its first pick, just press Enter. Otherwise, click on the one you want.

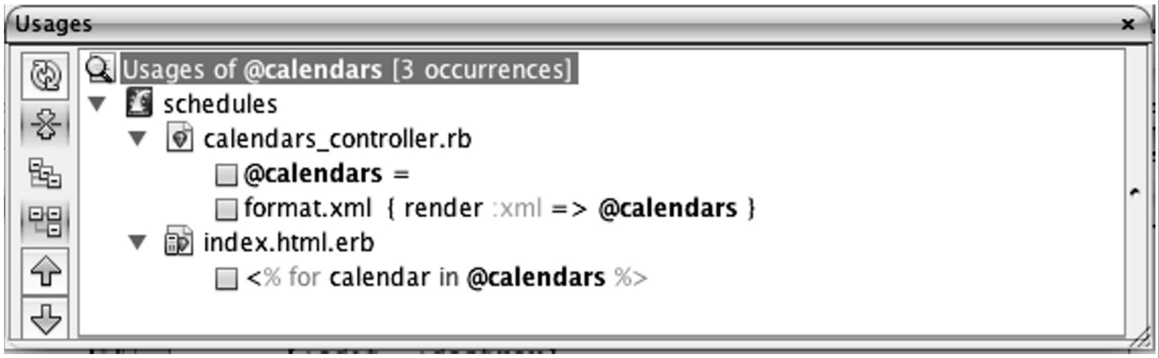
Navigate to the Next or Previous Occurrence

When the caret is on a variable or method call, the IDE highlights all occurrences of that member within the file and displays a yellow stripe for each occurrence in the right margin. Press Alt+Up Arrow or Alt+Down Arrow (Ctrl+Up Arrow and Ctrl+Down Arrow on the Mac) to jump from one occurrence to the next. Alternatively, click a yellow stripe in the right margin.

Find Usages

Right-click on a variable or method and choose Find Usages to see a list of all the usages in the project (see Figure 6-9). Double-click on a node in the list to jump to that usage. Click the Previous Occurrence and Next Occurrence buttons to jump from one usage to the next.

Figure 6-9. Usages Output



```
Usages of @calendars [3 occurrences]
▼ schedules
    ▼ calendars_controller.rb
        @calendars =
        format.xml { render :xml => @calendars }
    ▼ index.html.erb
        <% for calendar in @calendars %>
```

Note When you use Find Usages on a class, you can choose to find the class's subtypes instead of usages.

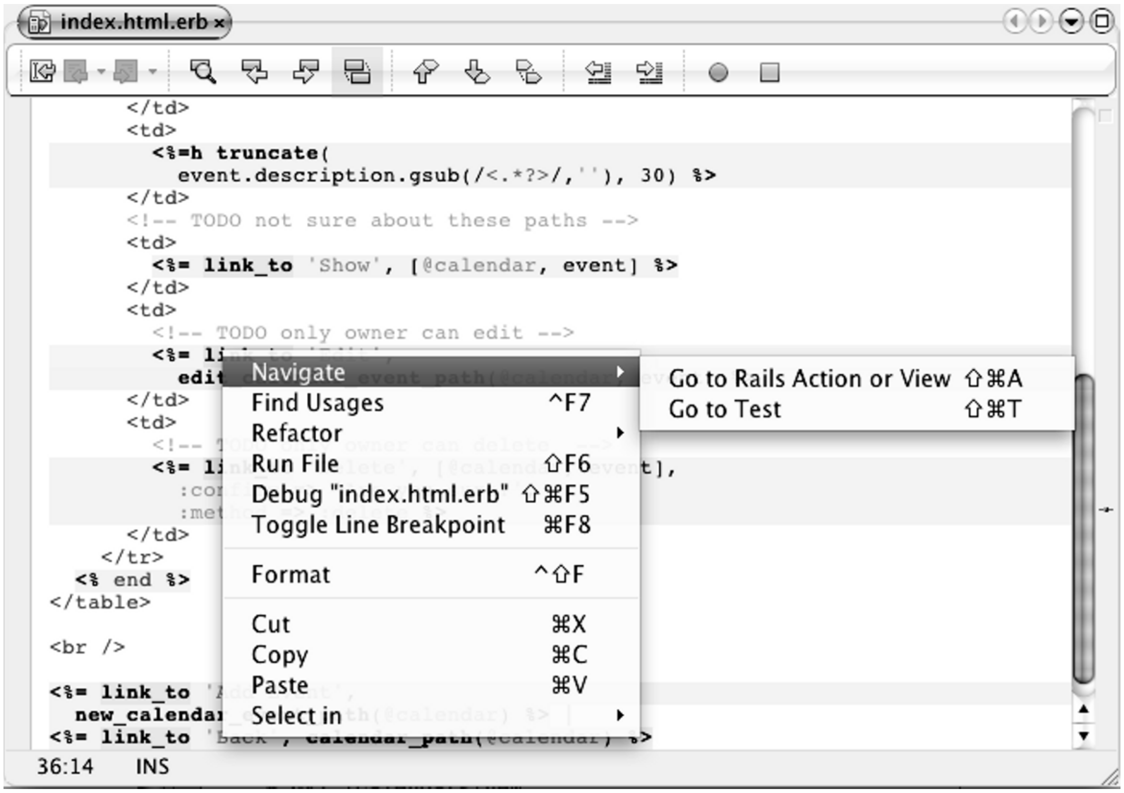
Navigating to Files

When working within the IDE, the files that you need are just a click or two away. In addition to the context-sensitive Go to Declaration menu action you learned about earlier, you can also Go to Test from most file types. With controller and view files, the pop-up menu has Go to Action or View, and with test files, the pop-up menu offers Go to Tested Class.

Note If both a unit test and an RSpec test exist for a file, the IDE takes you to the unit test.

In RHTML and ERB files you can Ctrl+Click on a partial argument in a `render` call to open the partial, and you can Ctrl+Click on a controller or action argument in a `link_to` call to open the controller file, as shown in Figure 6-10.

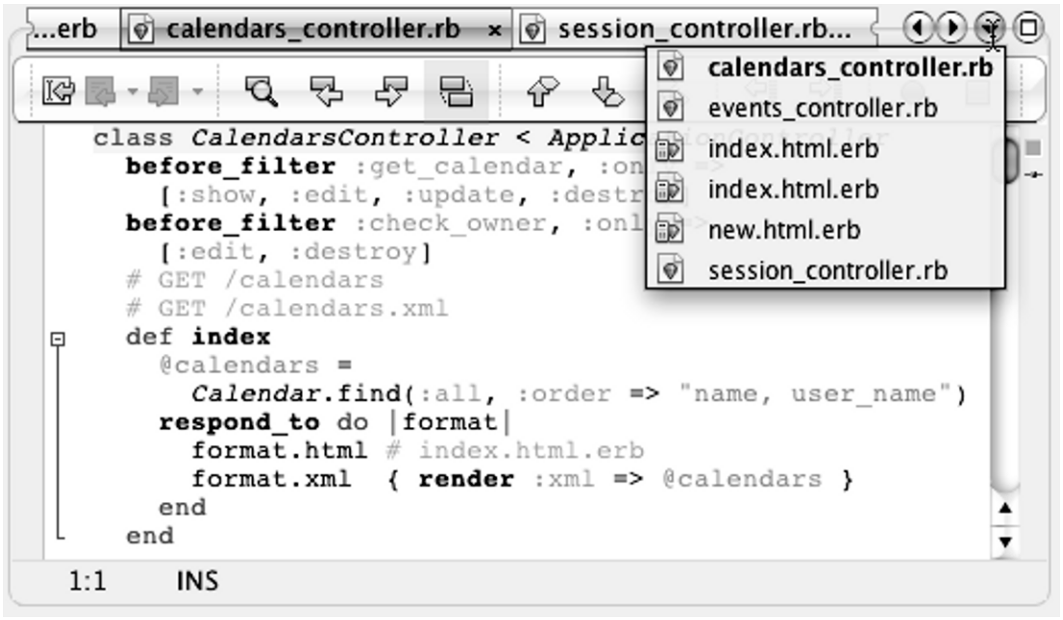
Figure 6-10. Navigating from a link_to to a Controller Action



```
        </td>
        <td>
          <%=h truncate(
            event.description.gsub(/<.*?>/,''), 30) %>
        </td>
        <!-- TODO not sure about these paths -->
        <td>
          <%= link_to 'Show', [@calendar, event] %>
        </td>
        <td>
          <!-- TODO only owner can edit -->
          <%= li
            edit
        </td>
        <td>
          <!--                                      t],
          <%= li
            :co
            :me
        </td>
      </tr>
    <% end %>
</table>

<br />

<%= link_to
  new_calenda
<%= link_to
```

You will usually have several files open in the editor. You can easily switch from one open file to the next by pressing Ctrl+Tab. When the list of open documents appears, press Ctrl+Tab repeatedly to go down the list to select the file to display. You can also display a file by selecting it from the open documents list (see Figure 6-11). Last, pressing Shift+F4 opens up a Documents dialog box that enables you to view, switch to, save, and close open documents.

Figure 6-11. Open Documents List



```
class CalendarsController < Applic
  before_filter :get_calendar, :on
    [:show, :edit, :update, :destr
  before_filter :check_owner, :onl
    [:edit, :destroy]
  # GET /calendars
  # GET /calendars.xml
  def index
    @calendars =
      Calendar.find(:all, :order => "name, user_name")
    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @calendars }
    end
  end
```

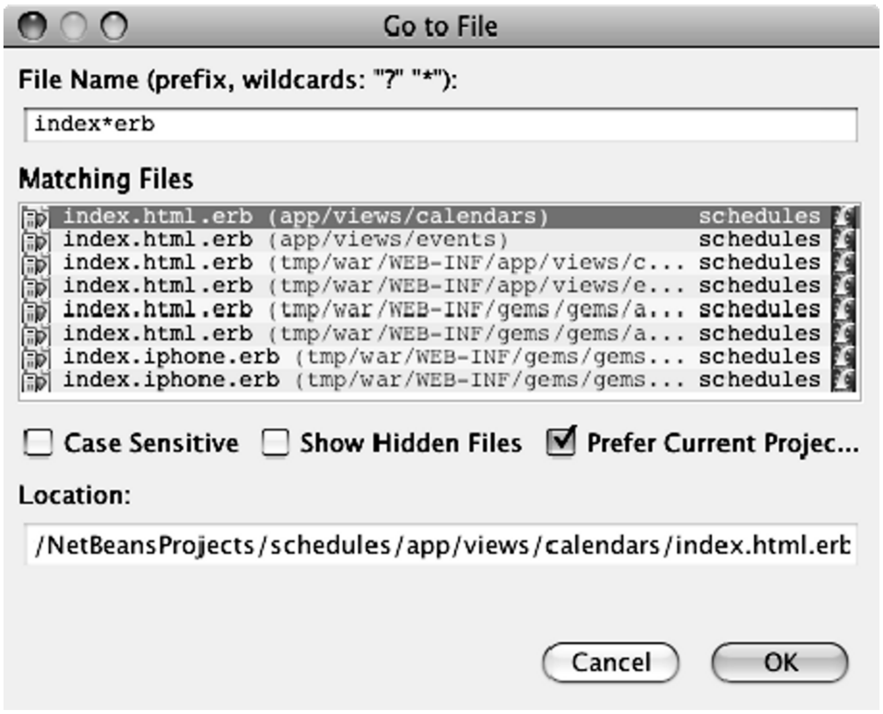
Tip If you have more files open than there is room for their tabs, use the Scroll Documents Left and Scroll Documents Right buttons in the editing tool bar to see the tabs for the other open files.

The IDE has four different search mechanisms for finding and opening files—Go to File, Find in Projects, Go to Type, and Go to Symbol—which we describe next.

Go to File

To quickly open a file, press Alt+Shift+O (Cmd+Shift+O on the Mac) to display the Go to File dialog box (see Figure 6-12). Type in the first few characters of the file's name to display a list of matching files. As you type, the list narrows. You can use the * (multiple characters) and ? (single character) wildcards to find files with names that contain a string or have a particular suffix. For example you can search for `index*erb` to find all the index views. The IDE searches all open projects, so it helps to select the Prefer Current Project check box to have the IDE list the files in the current project first.

Figure 6-12. Go to File Dialog Box


Go to File

File Name (prefix, wildcards: "?" "*"):

```
index*erb
```

Matching Files

```
index.html.erb (app/views/calendars)                   schedules
index.html.erb (app/views/events)                      schedules
index.html.erb (tmp/war/WEB-INF/app/views/c...         schedules
index.html.erb (tmp/war/WEB-INF/app/views/e...         schedules
index.html.erb (tmp/war/WEB-INF/gems/gems/a...         schedules
index.html.erb (tmp/war/WEB-INF/gems/gems/a...         schedules
index.iphone.erb (tmp/war/WEB-INF/gems/gems...         schedules
index.iphone.erb (tmp/war/WEB-INF/gems/gems...         schedules
```

☐ Case Sensitive ☐ Show Hidden Files ☑ Prefer Current Projec...

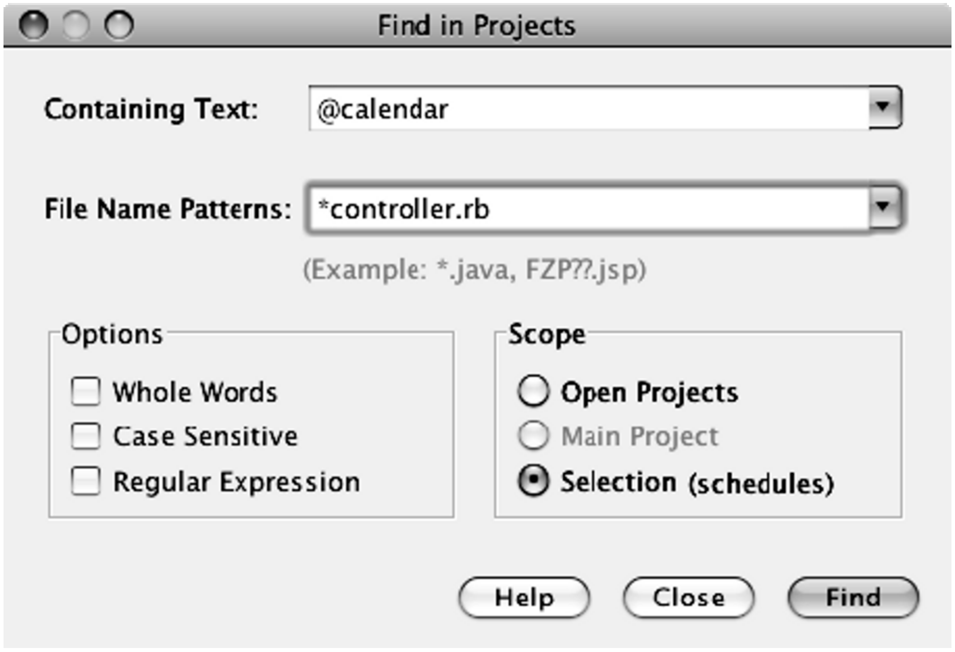
Location:

```
/NetBeansProjects/schedules/app/views/calendars/index.html.erb
```

Find in Projects

The Find in Projects dialog box is another way to find and open files (see Figure 6-13). Right-click the project's node, and choose Find from the pop-up menu. As with Go to File, you can use the * and ? wildcards to find all files with names that match a pattern. In addition, you can look for files that contain some specified text. Alternatively, you can use both text boxes to search for a string in the files that matches the file name pattern, such as looking for `@calendar` in all `*.rb` files. Use the Scope radio buttons to limit the search to the current project or to extend the search to all open projects.

Figure 6-13. Find in Projects Dialog Box



Containing Text: @calendar

File Name Patterns: *controller.rb

(Example: *.java, FZP??.jsp)

Options

- [ ] Whole Words
- [ ] Case Sensitive
- [ ] Regular Expression

Scope

Open Projects
Main Project
[✓ Selection (schedules)]

Tip If you do not know where an open file resides in the project structure, right-click in the source and choose Select in ➤ Projects from the pop-up menu. The IDE displays the file's node in the Projects window. You can also choose Select in ➤ Files to see the file in the Files window.

Go to Type and Go to Symbol

To find the file where a method, class, or module is defined, press Ctrl+O (Cmd+O on the Mac) to open the Go to Type dialog box (see Figure 6-14). You can use the following shortcuts to search for patterns:

- The * (multiple characters) and ? (single character) wildcards.

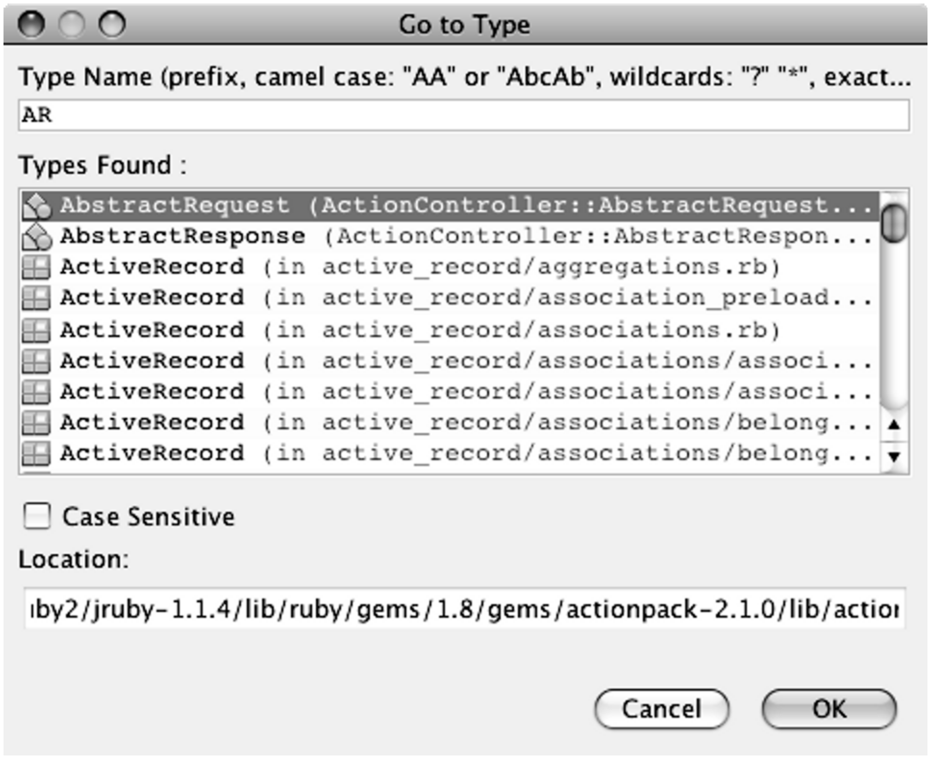
- Camel case shorthand. For example, use `AR` to find ActiveRequest, ActiveResponse, and ActiveRecord, and use `AC::B` to find ActionController::Base.
- # for methods. For example, `#to_x` finds all the `to_x` methods, and `A#to_x` finds all the `to_x` methods in modules that begin with A.

Double-click on a node in the results to open that file.

Tip Another way to quickly access the file is to type the method, class, or module name into the Quick Search box in the main tool bar.

You can find the files in which a symbol is defined by pressing Ctrl+Alt+Shift+O (Ctrl+Shift+Cmd+O on the Mac). This shortcut opens a Go to Symbol dialog box. Go to Symbol works across languages and searches for elements that are not normally considered to be types, such as JavaScript functions and Ruby methods. Just as with the Go to Type dialog box, you can use wildcards and camel case shorthand. Unlike Go to Type, you do not need to use the # character to search for a method.

Figure 6-14. Go to Type Dialog Box



Type Name (prefix, camel case: "AA" or "AbcAb", wildcards: "?" "*", exact...

AR

Types Found :

```
AbstractRequest (ActionController::AbstractRequest...
AbstractResponse (ActionController::AbstractRespon...
ActiveRecord (in active_record/aggregations.rb)
ActiveRecord (in active_record/association_preload...
ActiveRecord (in active_record/associations.rb)
ActiveRecord (in active_record/associations/associ...
ActiveRecord (in active_record/associations/associ...
ActiveRecord (in active_record/associations/belong...
ActiveRecord (in active_record/associations/belong...
```

Case Sensitive

Location:

ıby2/jruby-1.1.4/lib/ruby/gems/1.8/gems/actionpack-2.1.0/lib/actioı

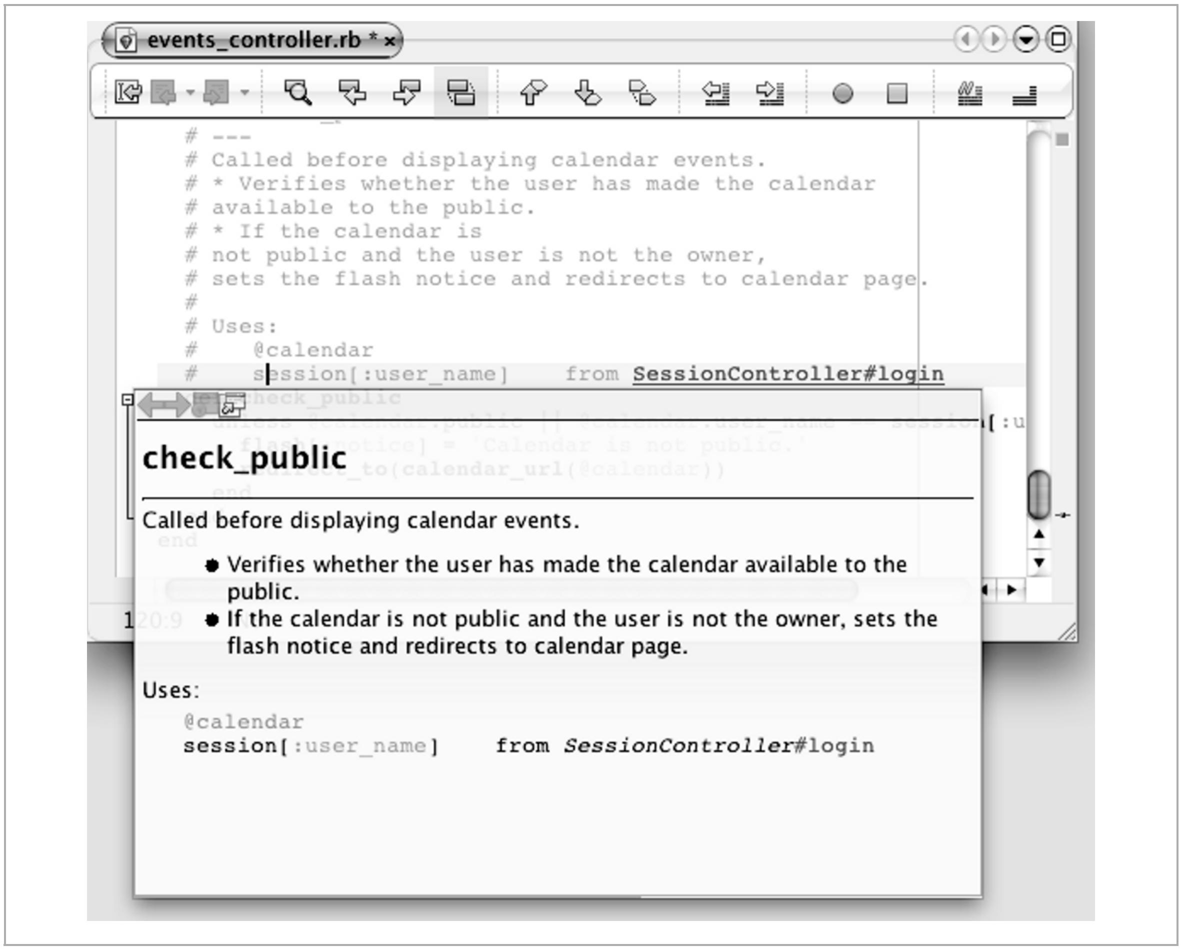
Tip If you can't remember the shortcuts for Go to Type and Go to Symbol, simply type **go to** into the Quick Search box in the main toolbar.

Viewing RDoc

When you use the code completion mechanism, you often see a display of the RDoc. You can also view the RDoc by placing the caret on the element that you want to look up and then pressing Ctrl+Shift+Space (Cmd+Shift +Space on the Mac).

If you want to see how your code comments will appear in RDoc format, just place the caret in your comments and use this same key sequence to see how your comments will be displayed in RDoc format, as shown in Figure 6-15. This helps to spot errors in the comments.

Figure 6-15. Displaying Comments as RDoc



```
# ---
# Called before displaying calendar events.
# * Verifies whether the user has made the calendar
# available to the public.
# * If the calendar is
# not public and the user is not the owner,
# sets the flash notice and redirects to calendar page.
#
# Uses:
#    @calendar
#    session[:user_name]    from SessionController#login
```

check_public

Called before displaying calendar events.

- Verifies whether the user has made the calendar available to the public.
- If the calendar is not public and the user is not the owner, sets the flash notice and redirects to calendar page.

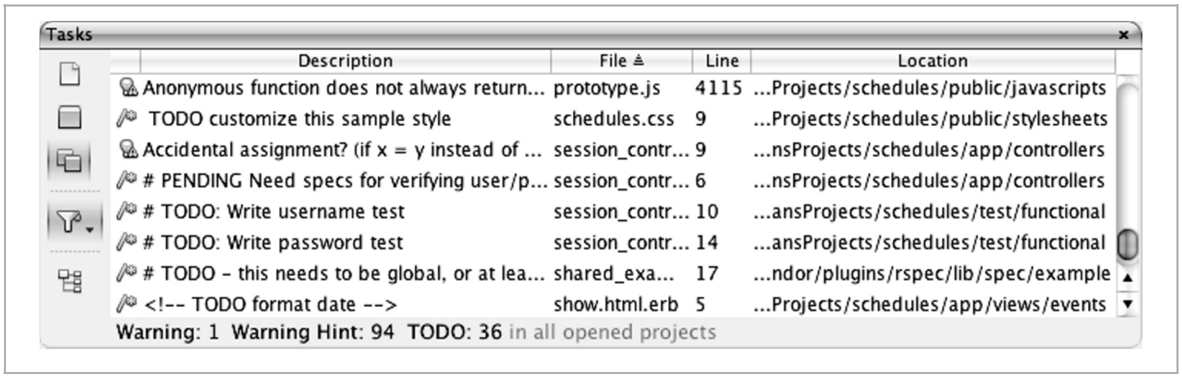
Uses:

```
@calendar
session[:user_name]    from SessionController#login
```

Working with Task Lists

So far, you have been learning how to find and fix code suggestions and errors on a file-by-file basis. The IDE has a Task List feature, which is very useful for getting an overview of all the hints and errors in a file, a project, or a set of projects (see Figure 6-16). To open the Task List window, choose Window ➤ Tasks from the main menu.

Figure 6-16. Task List Window



Description	File ≜	Line	Location
Anonymous function does not always return...	prototype.js	4115	...Projects/schedules/public/javascripts
TODO customize this sample style	schedules.css	9	...Projects/schedules/public/stylesheets
Accidental assignment? (if x = y instead of ...	session_contr...	9	...nsProjects/schedules/app/controllers
# PENDING Need specs for verifying user/p...	session_contr...	6	...nsProjects/schedules/app/controllers
# TODO: Write username test	session_contr...	10	...ansProjects/schedules/test/functional
# TODO: Write password test	session_contr...	14	...ansProjects/schedules/test/functional
# TODO - this needs to be global, or at lea...	shared_exa...	17	...ndor/plugins/rspec/lib/spec/example
<!-- TODO format date -->	show.html.erb	5	...Projects/schedules/app/views/events

Warning: 1 Warning Hint: 94 TODO: 36 in all opened projects

Use the top three buttons in the left margin to choose the scope of the list: file, project, or all projects. Use the Filter button's drop-down list to either show all types of tasks or filter out certain types. The bottom button toggles whether or not to group the items by type. To change the order of the items in the list, click in a column header. You can also right-click in the window and choose Sort By to sort the list by description, file, line, or location in ascending or descending order.

In addition to errors, warnings, hints, and quick fixes, you can have the list show all the comments that contain the text `TODO`, `@todo`, `FIXME`, `XXX`, `PENDING`, or `<<<<<<<`. This is useful for making notes to yourself about changes or fixes that you want to make. If you want to include other types of comments, open the Options manager by choosing Tools ➤ Options from the main menu. Click Miscellaneous, click the ToDo Tasks list, and click Add to add another pattern.

You can walk through the list and view the code for each entry by pressing Ctrl+Period (Cmd+Period for the Mac). You can also view the code by double-clicking an entry in the list. In both cases, the IDE places the caret in the editor location, so you can immediately press Alt+Enter to show the quick fixes, if there are any.

Viewing Recent Changes

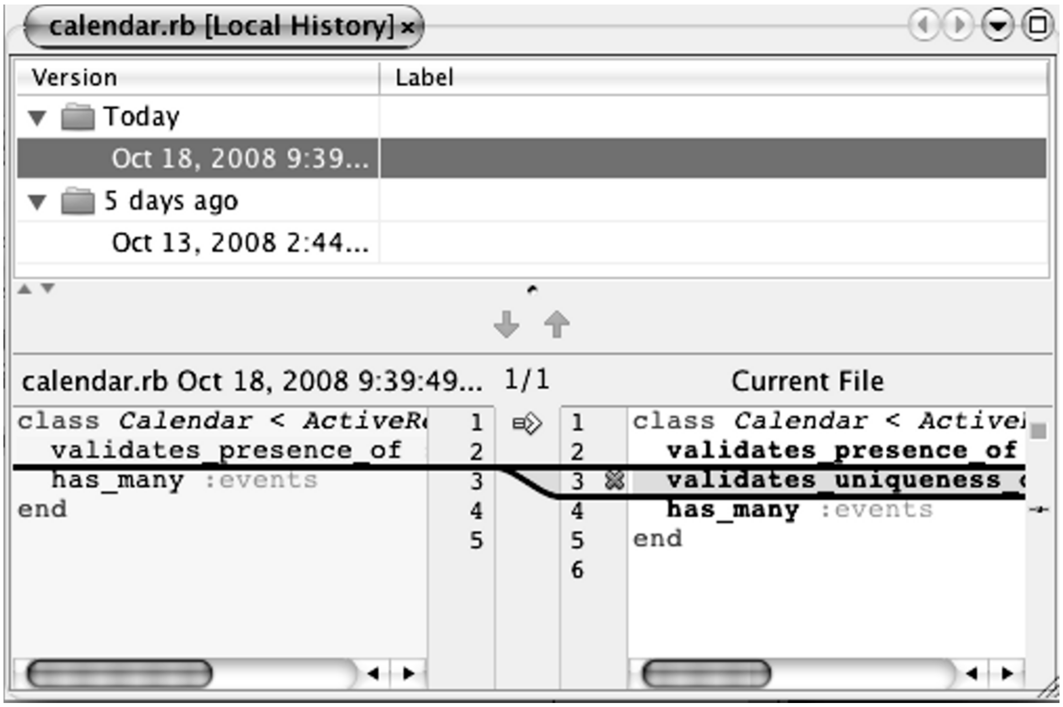
Even if you are not using source control, you can use the versioning tools to view the recent changes to your files. This can come in handy when the code stops working and you want to see what you recently did to break the code. When you edit your code, the IDE stores recent versions.

To view an earlier version, right-click on the file's node in the Projects window and choose Local History ➤ Show Local History. A Local History tab appears in the editor. You can select a version to view the differences between that version and the current version (see Figure 6-17), or you can right-click on an entry and choose either Revert from History or Delete from History from the pop-up menu. If you want, you can label each version. That enables you to use the pop-up menu item to show only rows where the label matches a filter.

You can also use the line-item buttons that appear in the comparison window to insert a deleted line or remove an added one. Click the arrow button that is between the old and new versions to restore the file to the old version.

If you have two versions of the same file you can have the IDE display a line-by-line comparison of the files. In the editor, right-click the file's tab and choose Diff To from the pop-up menu. Alternatively, in the Projects, Files, or Favorites window, use Ctrl+Click to select the two files and then choose Tools ➤ Diff from the main menu to display the differences.

Figure 6-17. Local History Tab



```
class Calendar < ActiveR
  validates_presence_of
  has_many :events
end
```

```
class Calendar < Active
  validates_presence_of
  validates_uniqueness_
  has_many :events
end
```

Keyboard Shortcuts

If you choose Help ➤ Keyboard Shortcuts Card from the main menu, you can get a PDF file that shows the most commonly used shortcuts. More complete lists can be found by choosing Help ➤ Help Contents, clicking the Search tab, and then searching on **keyboard shortcuts**. You will see shortcut lists for all types of files and windows. On the NetBeans Ruby Community wiki at `http://wiki.netbeans.org/RubyShortcuts` is a list that is maintained by the engineers. You can learn many of these shortcuts by looking at the menu actions, which show the keyboard shortcut equivalents for the actions. Table 6-5 shows some common shortcuts.

Table 6-5. Examples of Ruby Keyboard Shortcuts

ACTION	SHORTCUT	MAC SHORTCUT
Comment or uncomment	Ctrl+/	Cmd+/
Delete line	Ctrl+E	Cmd+E
Duplicate selection (above/below)	Ctrl+Shift+Up/Down	Cmd+Shift+Up/Down
Expand or complete the current word by inserting the next matching word from open buffers (hit repeatedly to cycle through matches)	Ctrl+K	Cmd+K
Indent or unindent	Tab/Shift+Tab	Tab/Shift+Tab
Open line under cursor	Shift+Enter	Shift+Enter
Open type (go to class)	Ctrl+O	Cmd+O
Open file by name prefix	Alt+Shift+O	Ctrl+Shift+O
Open Rails code generator (when cursor is in a `.rb` file)	Alt+Insert	Ctrl+Insert
Rename symbol under the caret (refactor)	Ctrl+R	Ctrl+R
Select the next enclosing block	Alt+Shift+Period	Ctrl+Shift+Period
Select progressively smaller blocks	Alt+Shift+comma	Ctrl+Shift+comma

Tip If you are familiar with Eclipse or Emacs keybindings, you can switch to those bindings by selecting Tools ➤ Options from the main menu, clicking Keymap, and selecting Eclipse or Emacs from the Profile drop-down list. There is a plug-in for vi from `http://jvi.sourceforge.net`.

Note Many of the editor shortcuts, such as Ctrl+Delete (Cmd+Delete on the Mac), work within word boundaries. By default, the editor considers an embedded capital letter as a word boundary. For example, if the cursor is in front of ActiveRecord, pressing Ctrl+Delete deletes Active but not Record. The same is true for underscores. If the cursor is in front of const_missing, Ctrl+Delete removes `const` and leaves `_missing`. See "Tweaking Under the Hood" in Chapter 10 for how to disable this feature.

Summary

Investing some time to learn the features that are provided by the NetBeans Ruby editor will greatly enhance your coding productivity. The live code assistance provides delimiter matching, error annotations, and code formatting. Semantic coloring gives you a clear visual picture of the elements that make up your code.

Context-sensitive code completion, including parameter tab completion, can help keep you focused on your code rather than having to turn to some external reference resource. In addition to code completion, live code templates alleviate the need to type boilerplate code over and over again. As with code completion, many of the live templates are parameterized (which is what makes them "live"). Finally, as you are coding, the NetBeans editor is constantly surveying your code for potential mistakes and suggesting improvements. Use the Task List to see all suggestions across one or more projects.

The IDE supports code refactoring in several ways. It enables you to easily rename elements, search and replace text, extract code into methods, and turn strings into constants or variables.

One of the advantages of using the IDE for your projects is that you can quickly and easily access all of a project's files. Not only do you have the Projects and Files windows to view your files in a hierarchical manner, but you have numerous options for finding and jumping to just the code you are looking for. When moving about a file you will want to take advantage of the Navigator window and the bookmarking feature. From within a file it is also possible to navigate to the declaration or find all usages of a class, method, or variable. Finally, there are dialogs to find files by name, type, or even its contents.

The local history feature allows you to easily compare, and if necessary, roll back to earlier versions of your code.

Getting familiar with the keyboard shortcuts (and not having to move your hand to the mouse) can really enhance your coding speed.

Finally, just about everything in this chapter is configurable—the formatting, colors, code hints, live templates, and keyboard shortcuts. Tailor the NetBeans editor to work in a way that makes you most productive.

Chapter 7: Testing and Debugging Projects

Test-driven development (TDD) promotes the writing of executable test cases before you start coding. After you add the desired functionality, you then run the tests to make sure the new functionality works as expected, and that the changes don't break something else. The NetBeans IDE provides quick access to the TDD support that is provided by Ruby and the Rails framework through the IDE's menu actions, keyboard shortcuts, and Test Results window.

When bugs are discovered, the IDE offers many tools to help you diagnose and resolve the problems. You can start up the integrated debugger to step through code, look at current values, watch for value changes, and view the call stack. To narrow in on your recent changes, which might have introduced the bug, you can open the Local History window to see the differences between recent versions. Last, you can easily view both the server and IDE log files in the Output window.

Creating Tests

The IDE automatically adds test folders when you create your projects. With Ruby projects you get a Test Files folder and an RSpec folder. With Ruby on Rails projects, you get folders for Unit Tests, Functional Tests, and Integration Tests. If the Rails project's Ruby platform has the RSpec gem in its repository, you also get the RSpec folder.

When you generate models, scaffolds, resources, and observers in Rails projects, the generator creates unit tests for you. For example, the controller generator creates a controller functional test, and the model generator creates a model unit test. Additionally, you can use the generator to create integration tests. If you have installed the RSpec and RSpec Rails plugins into your project, you can use the rspec_model, rspec_controller, and rspec_scaffold generators, which generate RSpec tests instead of unit tests. The plugin also adds `spec` tasks.

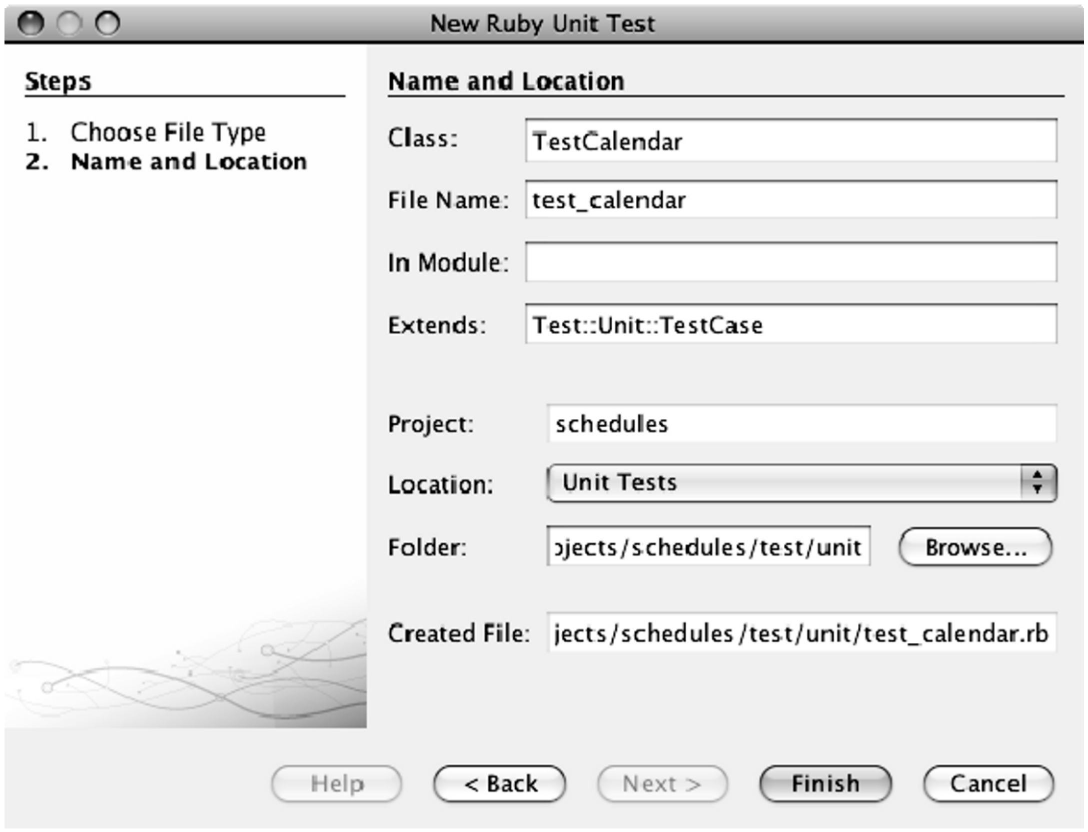
Tip The IDE associates your tests with the tested classes. You can quickly navigate from a class to its test by right-clicking in the source and choosing Navigate ➤ Go to Test, and you can navigate from a test to a class by choosing Navigate ➤ Go to Tested Class. If the class is associated with both a unit test and an RSpec class, the IDE takes you to the unit test.

Creating Unit Tests

The steps for creating unit tests are simple:

1. Right-click a project or folder node in the Projects window and choose New ➤ Ruby Unit Test from the pop-up menu to open the New Ruby Unit Test dialog box (see Figure 7-1).
2. Name the class, such as `TestCalendar`. The IDE derives the File Name from the class name (for example `test_calendar`), but you can change it. The File Name should begin or end with `test` so that it is recognized as a test.
3. You can specify a module and you can specify a different class to extend, such as ActiveSupport::TestCase, ActionController::TestCase, or ActionController::IntegrationTest.
4. If it is a Ruby project, choose Test Files from the Location drop-down list. For Ruby on Rails projects, you can select Unit Tests, Functional Tests, or Integration Tests.
5. If you want to put the test in a subfolder you can type it in or click Browse to navigate to the folder. You can also click Browse to add a subfolder.
6. Click Finish.

Figure 7-1. New Ruby Unit Test Dialog Box



Class: TestCalendar
File Name: test_calendar
In Module:
Extends: Test::Unit::TestCase
Project: schedules
Location: Unit Tests
Folder: ɔjects/schedules/test/unit
Created File: jects/schedules/test/unit/test_calendar.rb

The IDE seeds the class with a test method and with a call to `assert`, which causes the test to fail. It also provides the alternative `flunk` call, which prints out a reminder to write the test (see Listing 7-1). That way, you can stub out all your desired test classes and depend on the test results to remind you which tests you need to go back and write. Choose which of the two lines you want and delete the other. You might want to add a `TODO` comment so that the unfinished methods show up in the Tasks window.

Listing 7-1. Unit Test Class

```
# To change this template, choose Tools | Templates
# and open the template in the editor.

$:.unshift File.join(File.dirname(__FILE__),'..','lib')

require 'test/unit'
require 'calendar'

class Calendar < Test::Unit::TestCase
  def test_foo
    assert(false, 'Assertion was false.')
    flunk "TODO: Write test"
    # assert_equal("foo", bar)
  end
end
```

With Ruby projects you can also choose New ➤ Ruby Test Suite to create a test suite class (see Listing 7-2).

Listing 7-2. Test Suite Class

```
# To change this template, choose Tools | Templates
# and open the template in the editor.

require 'test/unit'

# Add your testcases here

require 'tc_1'
require 'tc_2'
require 'tc_3'
```

Tip If you use certain patterns in your test code, you can modify the template that the IDE uses to create the mock ups. See "Customizing Templates" in Chapter 10 for more information.

Adding Fixtures

Rails tests often depend upon fixtures for loading the test data. Fixtures can be in YAML or CSV format.

To add a YAML file, follow these steps:

1. Right-click a project or folder node and choose New ➤ Other from the pop-up menu.
2. Select the Ruby category, select YAML File from the File Types list, and click Next.
3. Name the file (without the `yml` suffix).
4. Specify the folder location and click Finish.

Because the IDE does not include support for CSV files, the steps are a bit different for that file type:

1. Right-click a project or folder node and choose New ➤ Other from the pop-up menu
2. Select the Other category.
3. Select Empty File from the File Types list and click Next.
4. Name the file (including the `csv` suffix).
5. Specify the folder location and click Finish.

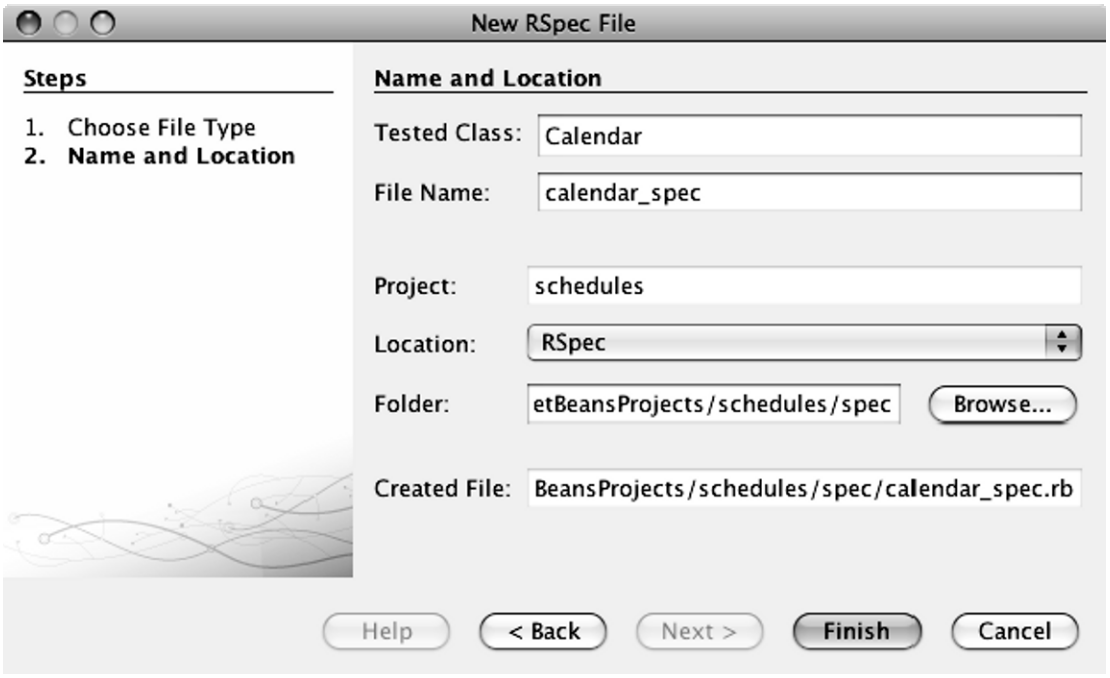
Creating RSpec Tests

RSpec is a framework that supports behavior-driven development (BDD) through specs and stories. If your project's platform includes this gem, the IDE offers RSpec support (the built-in JRuby platform that comes with the IDE includes the RSpec gem).

To create a spec, follow these steps.

1. Right-click a project or folder node in the Projects window and choose New ➤ RSpec file from the pop-up menu to open the New RSpec File dialog box (see Figure 7-2).

Figure 7-2. New RSpec File Dialog Box



New RSpec File

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Tested Class: Calendar

File Name: calendar_spec

Project: schedules

Location: RSpec

Folder: etBeansProjects/schedules/spec

Created File: BeansProjects/schedules/spec/calendar_spec.rb

2. Name the class you are testing. The IDE derives the File Name from the class name, but you can change it. You should keep the `_spec` suffix in the File Name so that it is recognized as a spec.
3. Choose RSpec from the Location drop-down list.
4. If you want to put the test in a subfolder, you can type it in or click Browse to navigate to the folder. You can also click Browse to add a subfolder.
5. Click Finish.

The IDE mocks up some executable code for the test and adds a `TODO` comment so that the Tasks window will display a reminder to write the test (see Listing 7-3).

Listing 7-3. Spec File

```
# To change this template, choose Tools | Templates
# and open the template in the editor.

require 'calendar'

describe Calendar do
  before(:each) do
    @calendar = Calendar.new
  end

  it "should desc" do
    # TODO
  end
end
```

Tip Just as with unit test classes, you can modify the template that the IDE uses to create the spec mock up. See "Customizing Templates" in Chapter 10 for more information.

As we mentioned in "Creating Tests," you can install the RSpec and RSpec Rails plugins and use the plugins' generators to create RSpec tests for you.

Note The NetBeans IDE 6.5 does not include support for RSpec stories.

Running Tests and Specs

You have two ways to run the tests in a specific test file:

- Right-click in a class's source and choose Test File from the pop-up menu.
- Right-click in the test's source and choose either Run File, Test File, or Debug "*test-file-name*" from the pop-up menu.

Tip Because the Test File action always runs the test, regardless of whether you are in the class source or the test source, you might want to get in the habit of using that action. The shortcut for Test File is Ctrl+F6 (Cmd+F6 on the Mac).

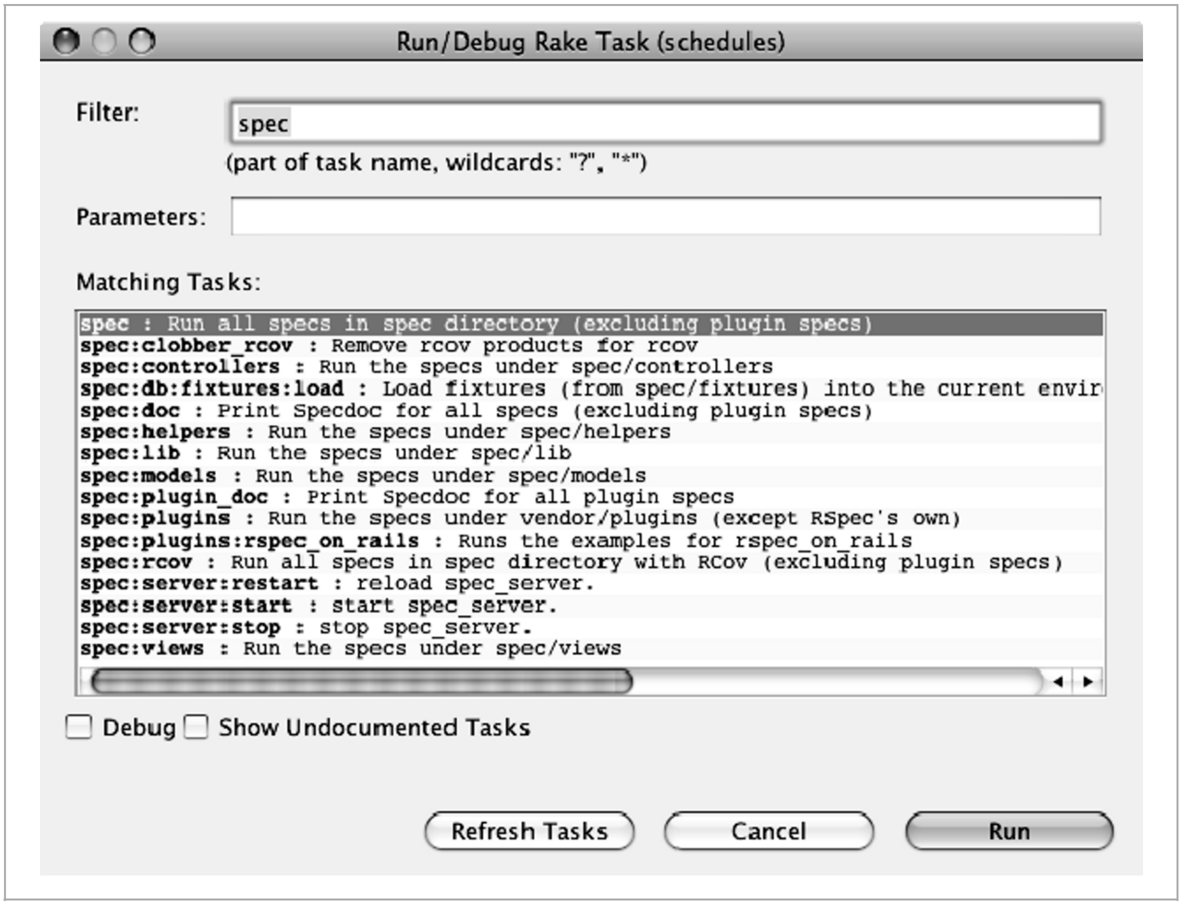
Although there are no menu actions available for this, you can click in a test file and run the specific test that the cursor is in. Press Alt+Shift+F6 to run the focused test and press Alt+Shift+F5 to debug it. Alternatively, type **test** in the quick search text box in the main toolbar, then choose Run Focused Test or Debug Focused Test from the search results.

To run all the tests for a project, simply right-click the project's node in the Projects window and choose Test for unit testing or choose RSpec Test for spec testing (RSpec Test is available only for projects for which the platform has the RSpec gem). This causes the IDE to invoke the corresponding Rake tasks (`test` tasks for the Test action and `spec` tasks for the RSpec Test action).

Note If your Rake files do not have the corresponding `test` or `spec` Rake tasks, the IDE runs all the test files that it finds under the test or spec test folders, whichever one applies. If you chose Test, it looks for file names that begin or end with `test`. If you chose RSpec Test, it looks for file names that end with `spec`.

You can also use the Run/Debug Rake Task window to run specific `test` and `spec` tasks. Right-click the project node and choose Run/Debug Rake Task. To see the available unit test tasks, type **test** in the Filter text box. Type **spec** in the Filter text box to see the available RSpec tasks (see Figure 7-3). Note that for Rails applications you must install the RSpec plugin in order to make the RSpec tasks available to the Rake tool.

Figure 7-3. RSpec Plugin Tasks in the Run/Debug Rake Task Window



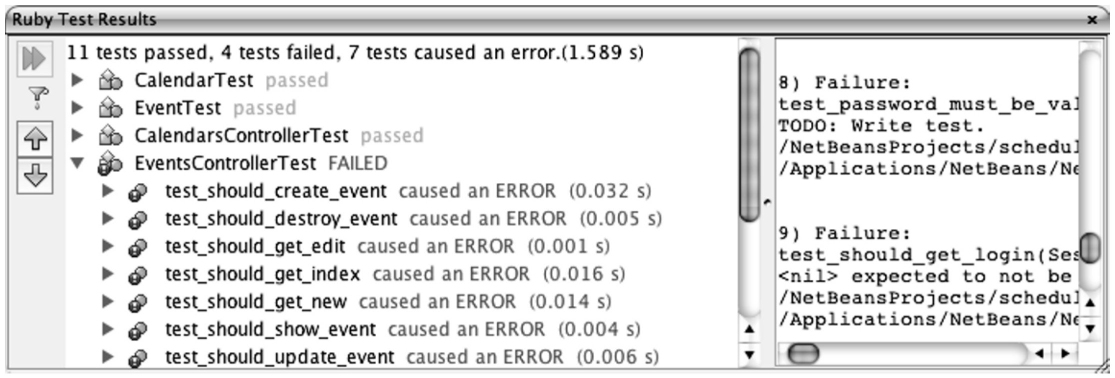
Filter: spec

```
spec : Run all specs in spec directory (excluding plugin specs)
spec:clobber_rcov : Remove rcov products for rcov
spec:controllers : Run the specs under spec/controllers
spec:db:fixtures:load : Load fixtures (from spec/fixtures) into the current envir
spec:doc : Print Specdoc for all specs (excluding plugin specs)
spec:helpers : Run the specs under spec/helpers
spec:lib : Run the specs under spec/lib
spec:models : Run the specs under spec/models
spec:plugin_doc : Print Specdoc for all plugin specs
spec:plugins : Run the specs under vendor/plugins (except RSpec's own)
spec:plugins:rspec_on_rails : Runs the examples for rspec_on_rails
spec:rcov : Run all specs in spec directory with RCov (excluding plugin specs)
spec:server:restart : reload spec_server.
spec:server:start : start spec_server.
spec:server:stop : stop spec_server.
spec:views : Run the specs under spec/views
```

When you run tests or specs, the IDE displays the output in Ruby Test Results window (see Figure 7-4). You can use the Previous Failure and

Next Failure buttons on the left side of the window to step through the failures and view the corresponding code in the source editor. Alternatively, you can double-click a test node or an entry in the stack in the left panel to open the file in the source editor and see the associated code.

Figure 7-4. Ruby Test Results Window



```
11 tests passed, 4 tests failed, 7 tests caused an error.(1.589 s)
▶ CalendarTest passed
▶ EventTest passed
▶ CalendarsControllerTest passed
▼ EventsControllerTest FAILED
    ▶ test_should_create_event caused an ERROR (0.032 s)
    ▶ test_should_destroy_event caused an ERROR (0.005 s)
    ▶ test_should_get_edit caused an ERROR (0.001 s)
    ▶ test_should_get_index caused an ERROR (0.016 s)
    ▶ test_should_get_new caused an ERROR (0.014 s)
    ▶ test_should_show_event caused an ERROR (0.004 s)
    ▶ test_should_update_event caused an ERROR (0.006 s)
```

```
8) Failure:
test_password_must_be_val
TODO: Write test.
/NetBeansProjects/schedul
/Applications/NetBeans/Ne


9) Failure:
test_should_get_login(Ses
<nil> expected to not be
/NetBeansProjects/schedul
/Applications/NetBeans/Ne
```

Tip Type **test** in the main toolbar's search text box for quick access to the test actions.

If you have Rake test task that doesn't follow the test or spec naming convention, and you want the IDE to display the task's results in the Test Results window, add lines similar to those in Listing 7-4 to either the project's `nbproject/project.properties` file (for shared properties) or the project's `nbproject/private/private.properties` file (for non-shared properties).

Listing 7-4. Test and Spec Task Properties

```
test.tasks=test,scene,role
spec.tasks=spec,contract
```

Tip The spec tasks obey the options that are specified in the file `spec/spec.opts`. To use a separate set of options when you run the tests from the IDE, put the options in a file named `spec/spec.opts.netbeans`. Because the default options in `spec.opts` might interfere with the Ruby Test Results window, it is a good idea to add a `spec.opts.netbeans` file that specifies the options that work best with the Ruby Test Results window. The IDE displays a warning about using `spec.opts` in the Ruby Test Results output window. To turn off the warning, pass the `-J-Druby.rspec.specopts.warn=false` option to the IDE, as shown in Chapter 10.

Using Autotest

If the project's platform has the ZenTest gem in its repository and you have set up your tests to meet the autotest specifications, you can right-click the project's node and choose AutoTest to start the autotester and open a Ruby Test Results window to display the autotester output. Then, whenever you edit a class or its test, the autotester runs the class's test.

Note At the time of this writing, autotest does not work with JRuby 1.1.4.

If you have the RSpec gem installed you need to have the correct configuration, as described here, in order for autotest to work:

RSpec 1.1.4. If the `spec` folder exists (RSpec Files in the Projects window), autotest runs the RSpec tests. If you want to run unit tests instead, you must either remove the `spec` folder or uninstall the RSpec gem.

RSpec 1.1.5 through 1.1.8. If you want autotest to run RSpec tests, create an `RSPEC` environment variable and set its value to `true`. One way to do this is to add the following line to the autotest script: `ENV['RSPEC'] = 'true'`.

If you are using RSpec tests, you should consult the RSpec and ZenTest web sites to verify whether you have compatible RSpec and ZenTest gems. As you would expect with Rails projects, your RSpec and RSpec Rails plugins must be compatible with the ZenTest version.

Tip If autotest does not appear to be working, choose Test or RSpec Test, whichever is applicable, to make sure that your tests are OK. Autotest's diagnostics are not as informative. Also, try running autotest from the command line to ensure that you have set up your autotest files correctly.

To stop the autotest process, click the X button that appears to the right of the AutoTest status at the bottom-right corner of the IDE, or click the X button in the left margin of the AutoTest tab in the Output window.

For more information about autotest, see `http://zentest.rubyforge.org/ZenTest/`.

Debugging Applications

Even when you plan for your application to cover all possible scenarios, write good tests to cover the functionality and usage, and carefully code to your tests, bugs are going to happen. Sometimes they even happen in the tests themselves. Finding the actual code that is causing the bug can be time consuming. The NetBeans debugger can often help you quickly get to the cause, make the changes, and verify that you have solved the problem.

Running the Debugger

Here are several ways to run the debugger:

- Click the Debug Project button.
- Press Ctrl+F5 (Ctrl+Cmd+F5 on the Mac).
- Choose Debug ➤ Debug Project from the main menu, or right-click the project's node and choose Debug.
- Open the class in the source editor, then right-click in the source and choose either Debug "*file-name*" to debug the class or choose Debug Test for "*file-name*" to debug the class's test. If you are in a controller class in a Rails application and right-click in a method, the debugger attempts to run the action. With Rails applications, you will often need to correct the URL that is sent to the browser, especially if you are using RESTful routes. If you are not sure what the correct URL is for the controller action or view, run the `routes` Rake task to find the correct URL for the action. Because the controller takes care of variables that the views need and often invokes essential methods from its filters, it is usually better to debug from the action than from the view.

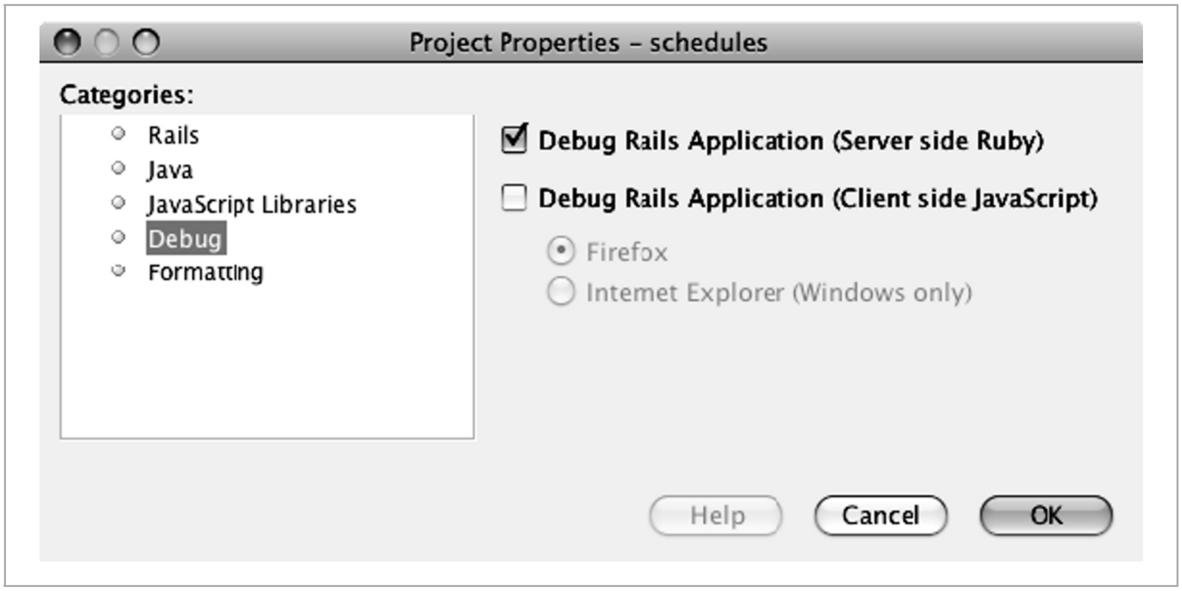
Note If you are using your own installation of JRuby and not the built-in JRuby, you must install the JRuby fast debugger manually. Go to `http://wiki.netbeans.org/RubyDebugging#section-RubyDebugging-JRuby` for installation instructions.

When you are in the development stage of a Rails application, you might find it best to start your application in debugger mode by clicking the Debug Project button and then just keep it running for the day. You can then do one of the following actions to get to the page you want to debug:

- Type the URL of the page you want to test into the browser's address text box
- Navigate through the application
- Use the browser's history list
- Click the browser's Reload, Back, or Forward button

When you start the debugger for a Rails project, a dialog box opens asking whether to perform client-side or server-side debugging. You can select the Do Not Show This Message Again check box to stop the dialog box from appearing for that project. If you do that, and then you later want to change the client-side or server-side setting, open the project's Project Properties dialog box, choose the Debug category, then clear the check box for client-side debugging (see Figure 7-5). Client-side debugging enables you to debug JavaScript. You can learn more about JavaScript debugging at `http://netbeans.org/kb/docs/web/js-debugger-ug.html`.

Figure 7-5. Debug Category in Project Properties Dialog Box

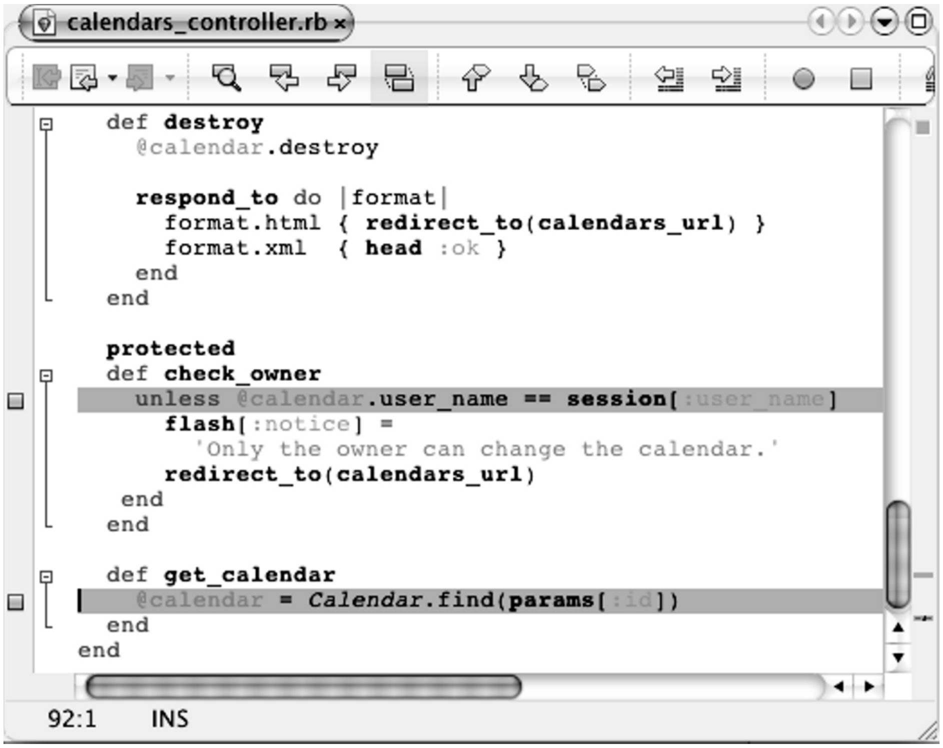


[✓] Debug Rails Application (Server side Ruby)
[ ] Debug Rails Application (Client side JavaScript)
[●] Firefox
[ ] Internet Explorer (Windows only)

Working with Breakpoints

Before you run your code in the debugger, you usually want to set breakpoints where you want the debugger to stop. To set a breakpoint, open the file, then click in the left margin next to the line that you want the debugger to stop on. You can also right-click the line and choose Toggle Line Breakpoint. A salmon-colored box appears in the left margin and the line is highlighted to indicate that a break point has been set. The right margin shows a salmon-colored line for each breakpoint. You can click those lines to jump to the corresponding breakpoint. Figure 7-6 shows a file with breakpoints.

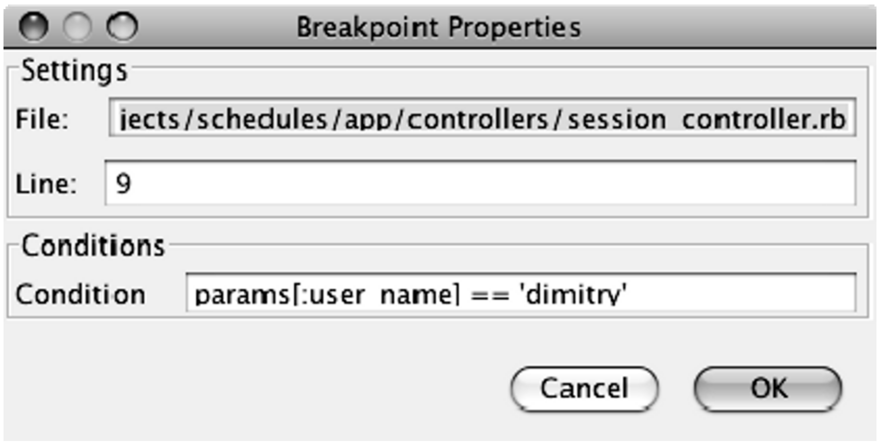
Figure 7-6. Line Breakpoints



```
  def destroy
    @calendar.destroy

    respond_to do |format|
      format.html { redirect_to(calendars_url) }
      format.xml  { head :ok }
    end
  end

  protected
  def check_owner
    unless @calendar.user_name == session[:user_name]
      flash[:notice] =
        'Only the owner can change the calendar.'
      redirect_to(calendars_url)
   end
  end

  def get_calendar
    @calendar = Calendar.find(params[:id])
  end
end
```

To create a conditional breakpoint, first create the breakpoint. Next, right-click the box that appears in the left margin along side the breakpoint, then choose Breakpoint ➤ Properties. Enter the condition, as shown in Figure 7-7. You can also turn a breakpoint into a conditional breakpoint in the Breakpoints tab of the Debugger window, which we discuss next, by right-clicking a breakpoint and choosing Customize.

Figure 7-7. Setting a Breakpoint Condition


Breakpoint Properties

Settings

File: jects/schedules/app/controllers/session_controller.rb

Line: 9

Conditions

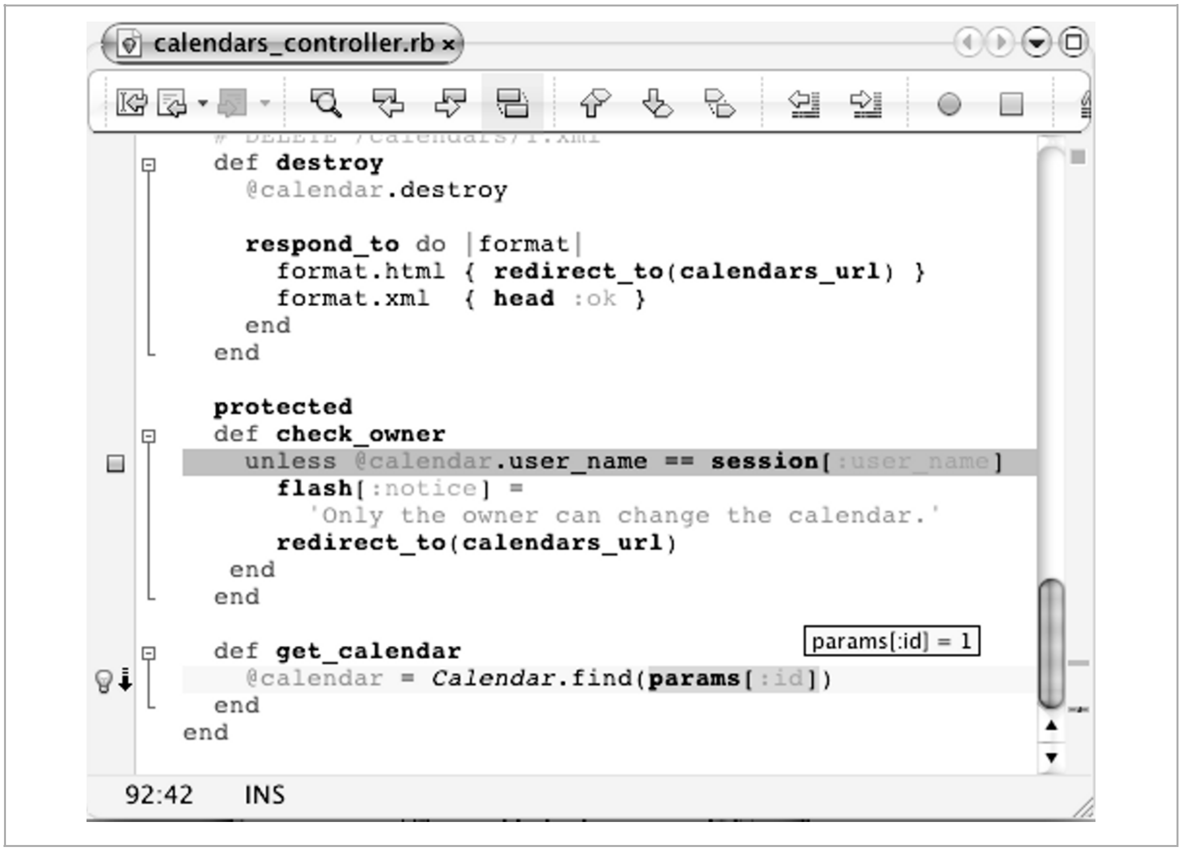
Condition params[:user_name] == 'dimitry'

You can also set exception breakpoints. You do that in the Breakpoints tab in the Debugger window, which we talk about in the next subsection.

When the debugger stops at a breakpoint, you can view the value of a variable or expression in the source by selecting it and then hovering the cursor over the selection until a tooltip appears showing the value (see Figure 7-8).

Tip You can also stop execution by clicking the Pause button in the main toolbar.

Figure 7-8. Viewing Values after Debugger Stops at a Breakpoint



```
  def destroy
    @calendar.destroy

    respond_to do |format|
      format.html { redirect_to(calendars_url) }
      format.xml  { head :ok }
    end
  end

  protected
  def check_owner
    unless @calendar.user_name == session[:user_name]
      flash[:notice] =
        'Only the owner can change the calendar.'
      redirect_to(calendars_url)
   end
  end

  def get_calendar
    @calendar = Calendar.find(params[:id])
  end
end
```

You can also use the tabs in the Debugger window, which we discuss next, to see the values of variables and watched expressions.

To continue executing the application, click one of the following buttons or choose Debug from the main menu, then choose the equivalent action:

Continue. Resume running the application and continue until the application reaches the next breakpoint or the application ends.

Step Over. Execute one source line. If the source line contains a call, execute the entire routine without stepping through the individual instructions.

Step Into. Execute one source line. If the source line contains a call, stop just before executing the first statement of the routine.

Step Out. Execute one source line. If the source line is part of a routine, execute the remaining lines of the routine and return control to the caller of the routine.

Run to Cursor. Run the current session to the cursor location in the editor, then pause the application.

To stop the debugging session, click the Finish Debugger Session button.

Tip If your application uses a database, you might want to open a query window on the database so you can view and monitor the database values while executing the code. First you need to register your database with the IDE, as shown in Chapter 3. Go to the Services window, expand Databases, and choose Connect. Then expand the node and expand Tables. Right-click on a Table node and choose View Data to open a SQL Command tab in the editor. You can repeatedly type in SQL commands and click the Run SQL button to view the results.

Using the Debugger Window

After you start the debugger, the Debugger window appears with the following four tabs (see Figure 7-9)

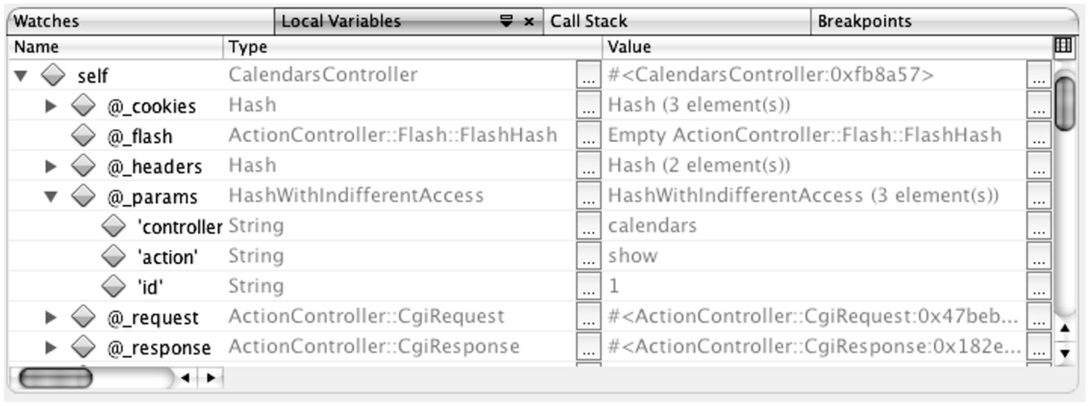
Watches. Lists all variables and expressions for which you have set watches. A watch enables you to track changes in the value of a variable or an expression during execution. To set a watch, open the source and select the variable or expression for which you want to set the watch, then right-click and choose New Watch.

Local Variables. Lists the local variables that are within the current call. You can also view the value of a variable or expression in the source by selecting it and then hovering the cursor over the selection. The IDE displays a tooltip, which shows the value.

Call Stack. Lists the sequence of calls made during the execution.

Breakpoints. Lists the breakpoints that have been set for all the open projects. You can disable and delete breakpoints in this tab. You can set exception breakpoints from this tab by right-clicking in the tab and choosing New Breakpoint. Choose Ruby from the Debugger drop-down list, enter the Exception Class name, and click OK.

Figure 7-9. Debugger Window



Name	Type	Value
self	CalendarsController	#<CalendarsController:0xfb8a57>
@_cookies	Hash	Hash (3 element(s))
@_flash	ActionController::Flash::FlashHash	Empty ActionController::Flash::FlashHash
@_headers	Hash	Hash (2 element(s))
@_params	HashWithIndifferentAccess	HashWithIndifferentAccess (3 element(s))
'controller	String	calendars
'action'	String	show
'id'	String	1
@_request	ActionController::CgiRequest	#<ActionController::CgiRequest:0x47beb...
@_response	ActionController::CgiResponse	#<ActionController::CgiResponse:0x182e...

You can optionally open the Sessions window by choosing Window ➤ Debugging ➤ Sessions from the main menu. The Sessions window is helpful for switching sessions when you are debugging more than one project at the same time. It is also helpful if you are debugging both JavaScript (client side) and Ruby (server side). You can stop all debugging sessions from this window by right-clicking in the window and then choosing Finish All. The Threads window is not applicable to Ruby in the 6.5 release. There is also a Sources window, which is helpful for client-side JavaScript debugging, but is not applicable to Ruby debugging.

Using the Local History Feature

In Chapter 6 you learned about the Local History feature. This is a great tool for finding bugs that have been introduced by recent changes. If you are adding new functionality and your tests start failing, right-click on the file’s node in the Projects window and choose Local History ➤ Show Local History. Select an entry and use the Up and Down keys to view the sequence of changes that you made to the file. Or perhaps revert to an earlier version and retest to see if the changes made after that caused the failures. You can also try out different solutions and use the Local History feature to revert to the original version after you finish testing the variations.

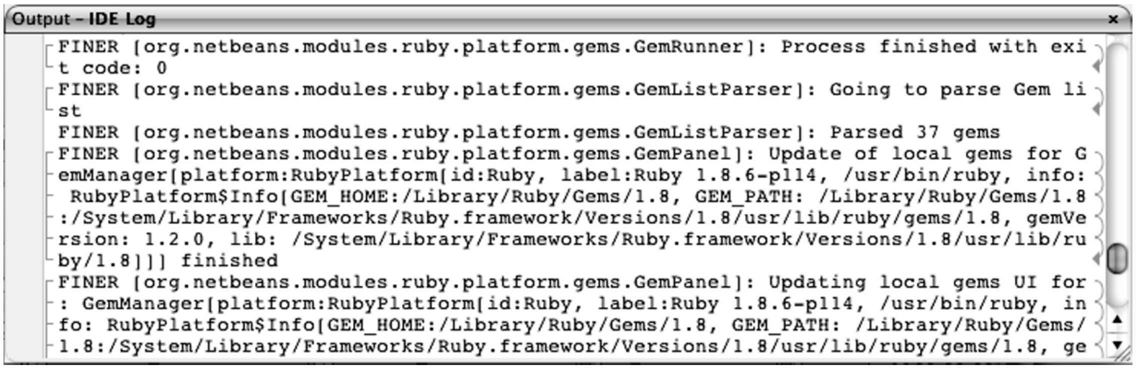
Viewing Log Files

With so many layers of software between you and the executing software, it is sometimes hard to determine if the bug is in your software, the IDE, the server, the Ruby platform, the Rails framework, a gem, a plugin, or the Rake software. Sometimes log files help locate the source of the problem.

When you run a Rails application, the IDE opens the server’s log file in the Output window. You can scroll through this log to get helpful information.

The IDE’s log file shows what it is doing in the back end and reports errors and warnings. To see the IDE’s log file, choose View ➤ IDE Log from the main menu. The log appears in the Output window as shown in Figure 7-10.

Figure 7-10. IDE Log Output



```
FINER [org.netbeans.modules.ruby.platform.gems.GemRunner]: Process finished with exi
t code: 0
FINER [org.netbeans.modules.ruby.platform.gems.GemListParser]: Going to parse Gem li
st
FINER [org.netbeans.modules.ruby.platform.gems.GemListParser]: Parsed 37 gems
FINER [org.netbeans.modules.ruby.platform.gems.GemPanel]: Update of local gems for G
emManager[platform:RubyPlatform[id:Ruby, label:Ruby 1.8.6-p114, /usr/bin/ruby, info:
 RubyPlatform$Info[GEM_HOME:/Library/Ruby/Gems/1.8, GEM_PATH: /Library/Ruby/Gems/1.8
:/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/gems/1.8, gemVe
rsion: 1.2.0, lib: /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ru
by/1.8]]] finished
FINER [org.netbeans.modules.ruby.platform.gems.GemPanel]: Updating local gems UI for
: GemManager[platform:RubyPlatform[id:Ruby, label:Ruby 1.8.6-p114, /usr/bin/ruby, in
fo: RubyPlatform$Info[GEM_HOME:/Library/Ruby/Gems/1.8, GEM_PATH: /Library/Ruby/Gems/
1.8:/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/gems/1.8, ge
```

If you enable detailed Ruby logging you can see the exact Ruby, JRuby, and Rails shell commands that the IDE invokes. You can then try to execute the command in a shell to see if the error is with the IDE or is due to some issue caused by the JRuby, Ruby, or Rake tool. See Chapter 10 to learn how to enable detailed logging.

Tip If the log output is too wide for the window, right-click in the window and choose Wrap Text from the pop-up menu. If you are only interested in seeing the output going forward, right-click and choose Clear.

Summary

The NetBeans IDE provides a UI that provides easy access to TDD and BDD support to help you develop quality code. In addition to Ruby's default testing framework, the IDE supports RSpec and autotest. In the cases where your tests (or your customers) do reveal a bug, you can use the integrated debugger to walk through your code and inspect your variables. The local history feature can be used to go "back in time" before a potential bug was introduced. In addition, easy access to log files helps you isolate at what layer things may be going wrong.

Chapter 8: Working with JRuby

The JRuby implementation of the Ruby language brings many benefits to the Ruby world. Topmost is the ability to leverage the Java technology stack. Java developers can use the rapid development features of Ruby and Rails without giving up their existing code base. Conversely, Ruby programmers can leverage Ruby and Rails functionality in Java applications though Java's scripting engine. Another feature that JRuby adds is native threading.

The JRuby implementation is quickly evolving as JRuby developers work to provide the best performance and memory utilization, as well make JRuby a great Java Virtual Machine (JVM) language.

The best place to begin learning about JRuby is at `http://wiki.jruby.org`. Here are three good places to start:

- `http://wiki.jruby.org/wiki/Getting_Started`
- `http://wiki.jruby.org/wiki/Java_Integration`
- `http://wiki.jruby.org/wiki/Calling_Java_from_JRuby`

Note The JRuby web site is in the process of moving to `http://jruby.kenai.com`.

Creating JRuby Projects

Creating a JRuby project is as simple as choosing a JRuby interpreter from the Ruby Platform drop-down list when you create a Ruby or Ruby on Rails application.

If you have an existing project that you want to switch to JRuby, do the following:

1. Right-click the project's node and choose Project Properties.
2. Select the Run category for Ruby projects, or select the Rails category for Ruby on Rails projects.
3. Select a JRuby entry from the Ruby Platform drop-down list.

When you create a JRuby Rails project you have several database configuration options. See the "Setting Database Configurations" section in Chapter 5 for more information about these options. If you are switching an existing project to use JRuby, you might want to consult that section to see what your `database.yml` configuration options are.

If a Rails project's database access is through a Java service layer, remember to edit the `config/environment.rb` file to remove Active Record from the frameworks configuration by uncommenting the following statement and removing the symbols that you do not want to turn off:

```
    # config.frameworks -= [ :active_record,
        :active_resource, :action_mailer ]
```

In addition, if you are using a Rails 2.1 version, which is the case with the bundled JRuby software, you must also comment out or remove the following Active Record references in `config/initializers/new_rails_defaults.rb`:

```
ActiveRecord::Base.include_root_in_json = true
ActiveRecord::Base.store_full_sti_class = true
```

If you have a Rails application that you intend to sometimes run using the JRuby interpreter and sometimes the CRuby interpreter, you might want to use code like the following in your `environment.rb` file:

```
 if defined? JRUBY_VERSION
   #do jruby stuff
 else
   #do cruby stuff
 end
```

Here is an example of modifying the `database.yml` file to use the JDBC MySQL adapter when running with JRuby:

```
<% jdbc = defined?(JRUBY_VERSION) ? 'jdbc' : '' %>
development:
  adapter: <%= jdbc %>mysql
```

One problem with switching back and forth is that you need to make the same set of gems available for both platforms. You can put the gems that are valid for both CRuby and JRuby (some gems that have native extensions don't work with JRuby) in a separate repository and use the Platform Manager to add that repository to the Gem Path for both platforms. If you use gems that have native extensions in your CRuby application, you might not be able to run the application using JRuby.

Tip If you need to run JRuby from the IDE using a different JDK than the IDE, use the `-Djruby.java.home` property to pass the path to the JDK to the IDE, as shown in this example: `-J-Djruby.java.home=/Java/Versions/1.5/Home`. See Chapter 10 to learn how to pass properties to the IDE.

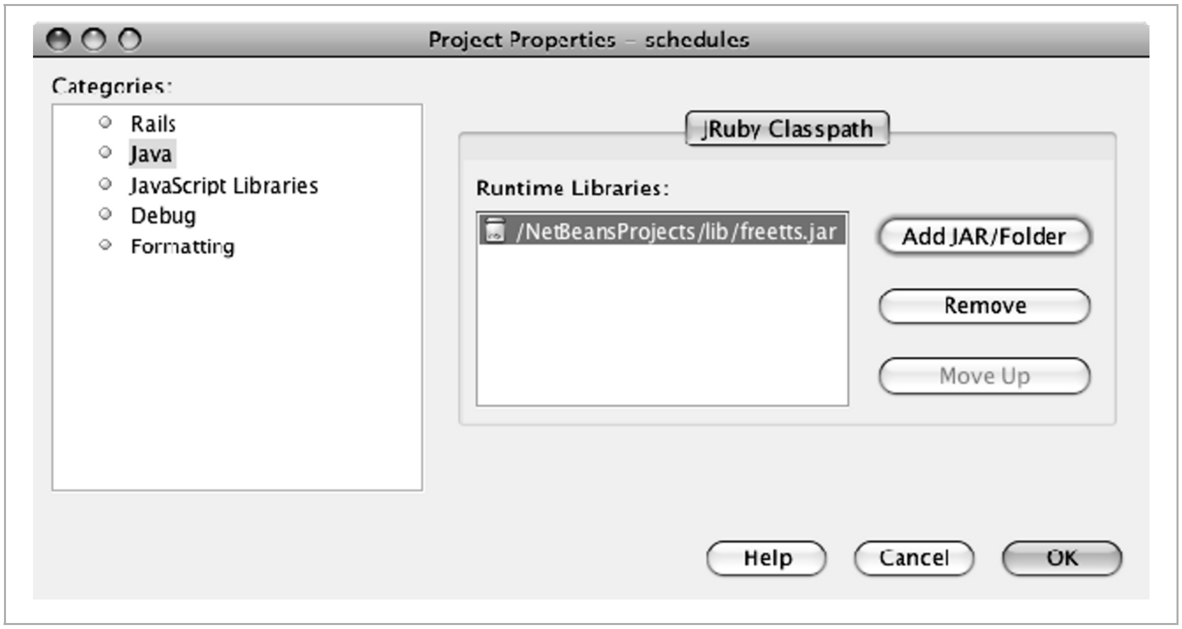
Adding Java Libraries to JRuby Projects

The standard Java classes are included with JRuby. To use third-party Java libraries or Java technologies such as Java Enterprise Edition (Java EE) in your JRuby projects, you must make them available to your application. The easiest way to do that is to use the Project Properties dialog box, as shown in the following steps, to add the JAR file to the classpath (see Figure 8-1):

1. Right-click the project's node and choose Project Properties from the pop-up menu.
2. Select Java from the Categories list.

3. Click Add JAR/Folder.
4. Navigate to and select the JAR file.
5. Click Open.

Figure 8-1. Adding JAR Files to a JRuby Project



Using Java Classes and Methods in JRuby Projects

There are many resources for learning how to use Java classes and methods in JRuby projects. At the beginning of this chapter we listed some places to get started. To get you going quickly we will introduce some of the basics, as of JRuby 1.1.4. Keep in mind that the Java integration with JRuby is rapidly changing, so this code might be out of date by the time you read this.

When you use Java in your Ruby class, the first thing you need to do is to add the following statement, which loads the Java support:

```
require 'java'
```

Just as with Java code, you can import the Java classes that you will use. If the class's top-level package is `com`, `org`, `java`, or `javax`, the import statement is just like a Java import statement.

```
import javax.swing.JFrame
```

For other Java classes you must put the fully-qualified class name in quotes, as shown in the following example:

```
import "myorg.schedule.Event"
```

Alternatively, you can skip the import statement and just use the fully-qualified class names in the format shown here:

```
myEvent = Java::myorg::schedule::Event.new()
label = Java::javax::swing::JLabel.new("Events")
```

If the top-level package is `com`, `org`, `java`, or `javax`, you can use this Java-like format:

```
label = javax.swing.JLabel.new("Events")
```

To include a package you must use a module. You then reference a class in the package by the module name, as shown in the following code example:

```
module JavaSwing
  include_package "javax.swing"
end
label = JavaSwing::JLabel.new()
```

Tip There are several package modules already available. For example you can access `javax.swing` classes by the `JavaxSwing` module, such as `Java::JavaxSwing::JLabel.new("Events")`. To see the module name for a Java class, type ***class*.ancestors** in a JRuby IRB console, for example **Java::javax::accessibility::Accessible.ancestors**.

JRuby enables you to use Ruby-like syntax when you reference a Java class's methods. For example you can call the `JFrame.setVisible` method using `setVisible`, `visible=`, or `set_visible`. To test if the frame is visible you can use `visible`, `visible?`, `is_visible?`, or `is_visible`. To find out the variations of a class's methods, open an IRB console and use the class's `methods` call to display the list of methods. Here is an example:

```
>>Java::javax::swing::JLabel.methods
=> ["get_default_locale", "defaultLocale=",
 "setDefaultLocale", "defaultLocale",
 "getDefaultLocale", …
```

Setting JRuby Runtime Properties

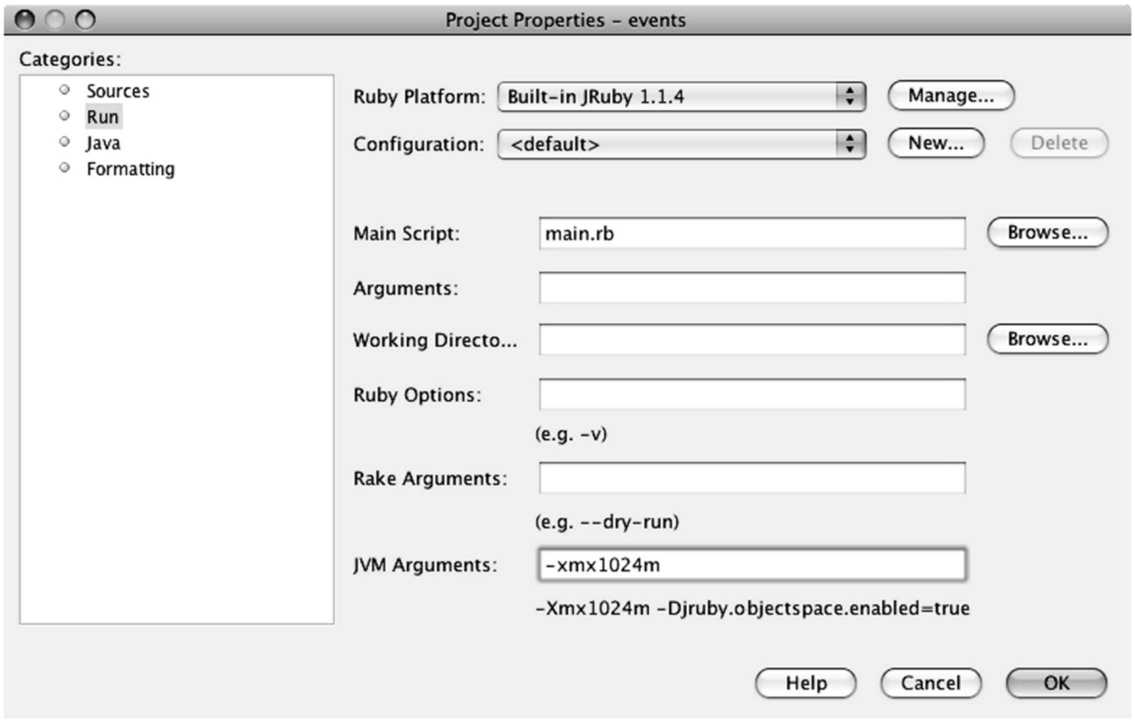
For Ruby projects you can pass options to the JVM using the JVM Arguments text box in the Project Properties dialog box (see Figure 8-2). In addition, you can use this text box to pass JRuby runtime properties for fine-tuning its performance or compatibility. To learn more about JVM parameters and runtime properties, go to `http://wiki.jruby.org/wiki/Performance_Tuning`. You can also see the list of runtime properties by typing **jruby –properties** in a command window.

Note Currently the UI only allows you to specify JVM properties for Ruby projects. This might change in a future release. In the meantime, if you want to pass a JVM argument in a Rails project, open the *project-dir*`/nbproject/project.properties` file and add a `jvm.args` property, such as `jvm.args=-Xmx=1024m`.

To set the properties, complete these steps:

1. Right-click the project's node and choose Project Properties from the pop-up menu.
2. Select Run from the Categories list.
3. Type the property settings in the JVM Arguments text box. Omit the leading `-J`. For example, for the property `–J-Xmx512m`, type **–Xmx512m**.

Figure 8-2. Setting JRuby JVM Properties



Project Properties - events

Categories:
- Sources
- Run
- Java
- Formatting

Ruby Platform: Built-in JRuby 1.1.4
Configuration: <default>
Main Script: main.rb
Arguments:
Working Directo...:
Ruby Options: (e.g. -v)
Rake Arguments: (e.g. --dry-run)
JVM Arguments: -xmx1024m
-Xmx1024m -Djruby.objectspace.enabled=true

Using JRuby in Java Projects

Should you want to leverage JRuby technology in a Java application, you have a few options for doing so:

JSR 223: Scripting for the Java Platform. JSR 223, which is included in the Java SE 6 platform release, provides script engine mechanisms for using scripting language programs in Java applications.

Bean Scripting Framework (BSF). The BSF, which is an Apache Jakarta Project, provides scripting language support in Java applications. JRuby offers Bean Scripting Framework (BSF) support in its `lib/bsf.jar` JAR file.

Direct JRuby Embedding. You can embed JRuby directly using the JRuby API. However, the JRuby wiki recommends that you use JSR 223 or BSF instead.

To learn more about these technologies, see the Java Integration wiki page at `http://wiki.jruby.org/wiki/Java_Integration`.

The following three code listings show an example of using the JSR 223 script engine mechanism to access JRuby. Listing 8-1 is a Ruby class that sets the `$current_date_string` global variable. Listing 8-2 is a Java class that evaluates the Ruby script and gets the value of `$current_date_string`. The main class in Listing 8-3 displays the results. This Java application requires two libraries: `jruby.jar`, which is in JRuby's `lib` folder, and `jruby-engine.jar`, which is available from scripting.dev.java.net.

Listing 8-1. time.rb

```
t = Time.now
$current_date_string = t.strftime("%A, %c")
```

Listing 8-2. RubyCurrentDateString.java

```
package rubyfromjava;

import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

public class RubyCurrentDateString {

  private ScriptEngine engine;
  private InputStream iStream;

  public void init() {
  }

  public Object eval() {
    // ScriptEngineManager object can tell you what
    // script engines are available to the JRE
    // It can also provide ScriptEngine objects
    // that interpret scripts written in a specific
    // scripting language.
    ScriptEngineManager manager =
      new ScriptEngineManager();
    // Get the script engine object that interprets
    // Ruby scripts
    // (requires jruby.jar and jruby-engine.jar)
    engine =
      manager.getEngineByName("jruby");
    Object currentDateString =
      new Object();
    ClassLoader loader =
      getClass().getClassLoader();
    iStream =
      loader.getResourceAsStream("scripts/time.rb");
    try {
      Reader reader = new InputStreamReader(iStream);
      engine.eval(reader);
      currentDateString =
        engine.getContext().
        getAttribute("current_date_string");
      return currentDateString;
    } catch (ScriptException ex) {
      ex.printStackTrace();
    }
    return currentDateString;
  }
}
```

Listing 8-3. Main.java

```
package rubyfromjava;

public class Main {

  public static void main(String[] args) {
    RubyCurrentDateString rubyScript =
            new RubyCurrentDateString();
    Object currentDateString = rubyScript.eval();
    System.out.println(
            currentDateString.toString());
  }
}
```

Note This example is written for version 6 of the JDK software. If you are using version 5, you need the JSR-223 API reference implementation from `http://www.jcp.org/en/jsr/detail?id=223`.

Unfortunately, the NetBeans source editor does not provide the Ruby editing features when you edit your Ruby code in a Java project, nor will you be able to test or run the Ruby code. You might want to create a separate Ruby project, from which you can open an IRB or Rails console, and from which you can test out your code. Alternatively, you can create a project for your Ruby code and provide symbolic links from the Java project to the scripts, or the other way around.

Summary

Part of the IDE's added value is that it comes bundled with the JRuby interpreter. Although JRuby is not directly related to the NetBeans IDE, this chapter introduced the basic JRuby concepts and how its unique features are supported by the IDE.

For most Ruby and Rails projects, running on the JRuby interpreter will be transparent to you. However, this chapter provides several recommendations for optimizing your environment when switching from Ruby to JRuby. Once your project is running on JRuby, you can easily add Java libraries to your project and begin using the Java classes from those libraries. JRuby even lets you call Java methods using the Ruby naming conventions that you are more familiar with.

Chapter 9: Deploying Rails Applications

At some point you might want to deploy your Rails application to a Java servlet container such as Tomcat or the GlassFish application server. For example, perhaps your deployment environment doesn't offer a Ruby container. Or perhaps you need better scaling.

The Warbler gem packages a Rails application into a WAR file for deployment to a Java servlet container. In this chapter we show how the IDE provides integration with Warbler, and how to generate and deploy the WAR file.

We just touch the surface on how to use Warbler. The best place to begin learning about Warbler is at `http://warbler.kenai.com`.

Creating WAR Files with the Warbler Plugin

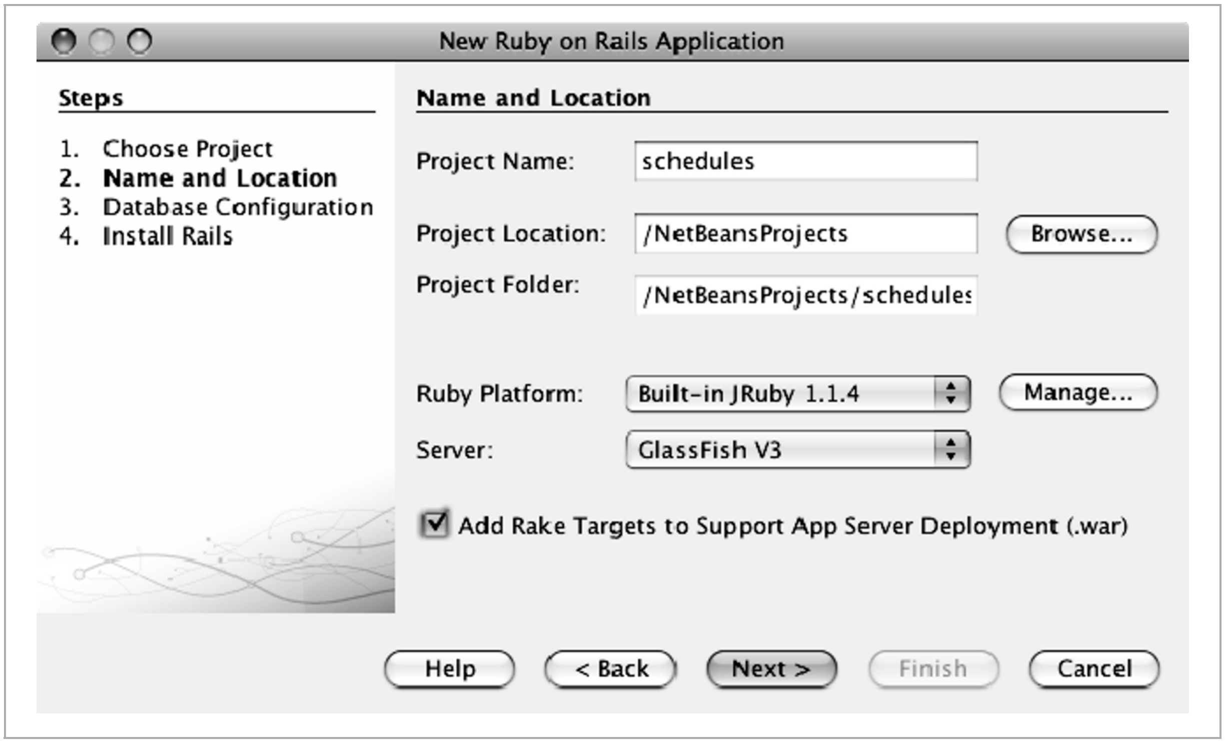
To use the Warbler gem most effectively from within NetBeans, you'll also want to install the Warbler plugin into your project. By installing the Warbler plugin, you will be able to use the IDE's Generator and Rake dialog boxes to create a WAR file, as we will show here. Without the Warbler plugin, you'll have to run the Warbler executable from the command line.

Note You need the JDK to use Warbler.

Installing the Warbler Plugin

The easiest way to install the Warbler plugin is during project creation. On the first page of the wizard is a check box labeled Add Rake Targets to Support App Server Deployment. Select this box to add the Warbler plugin (see Figure 9-1).

Figure 9-1. Add Rake Targets to Support App Server Deployment Check Box


New Ruby on Rails Application

Steps

1. Choose Project
2. **Name and Location**
3. Database Configuration
4. Install Rails

Name and Location

Project Name: schedules

Project Location: /NetBeansProjects

Project Folder: /NetBeansProjects/schedules

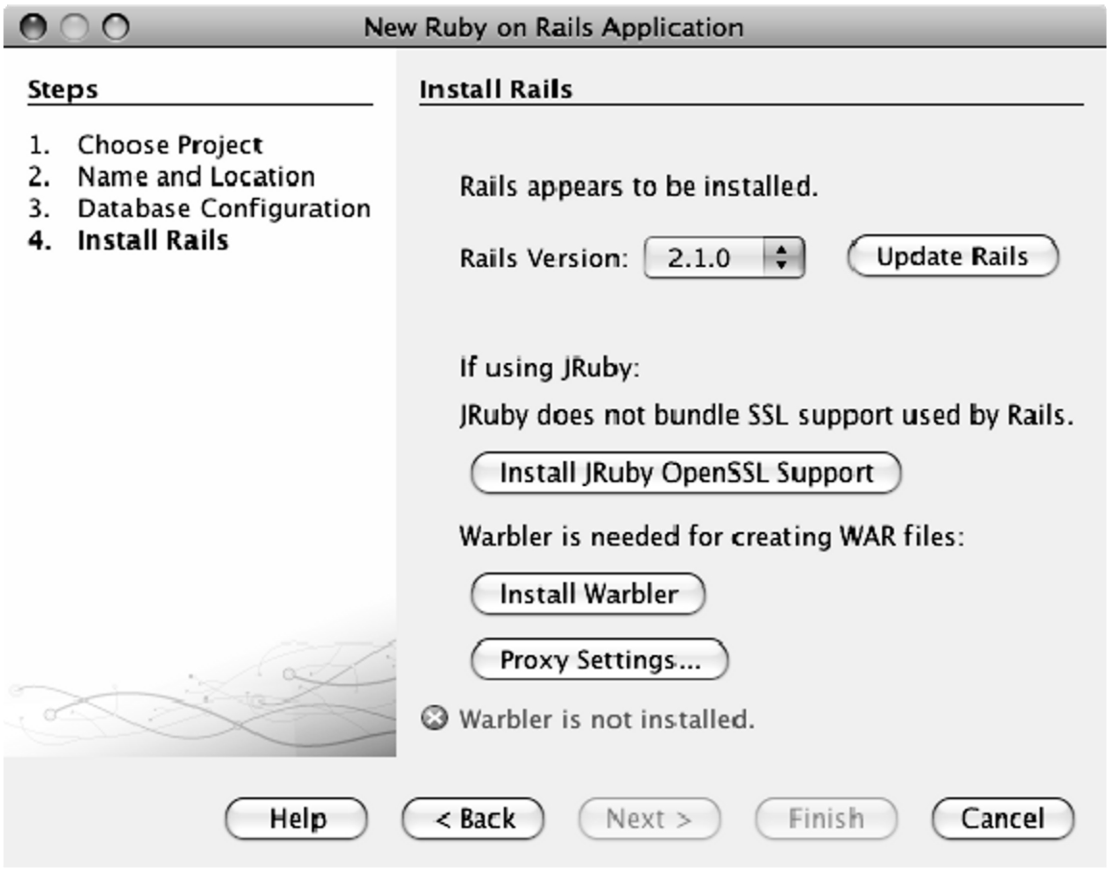
Ruby Platform: Built-in JRuby 1.1.4

Server: GlassFish V3

☑ Add Rake Targets to Support App Server Deployment (.war)

If you have not installed the Warbler gem for the project's Ruby platform, the IDE will force you through all the steps in the wizard. If you don't need to make changes in the intermediate steps, just click Next until you get to the last one. On the last step, click the Install Warbler gem and click Finish.

Figure 9-2. Install Warbler Button



Rails appears to be installed.

Rails Version: 2.1.0

If using JRuby:

JRuby does not bundle SSL support used by Rails.

Warbler is needed for creating WAR files:

Warbler is not installed.

If your project already exists, follow these steps to add the Warbler plugin to the project:

1. If you haven't already, add the Warbler gem to the Ruby platform's repository.
2. In a terminal window, change to the project folder.
3. Type the appropriate command and press Enter. If the command fails, verify that you have installed the Warbler gem into that Ruby or JRuby repository.

- `jruby -S warble pluginize`
- `ruby -S warble pluginize`

Tip The `jruby` executable may not be in your path environment variable. You can easily copy its full path from the Interpreter field of the Ruby Platforms dialog box.

When you use Warbler with your CRuby projects, keep in mind that the packaged WAR file uses JRuby. Therefore, your application cannot use gems or code that does not work with JRuby. We talked about some of the issues and workarounds in "Creating JRuby Projects" in Chapter 8.

If you are using a database other than MySQL, make sure that the JRuby platform has the necessary JDBC adapter gems installed, and that you are using a JDBC adapter in the production configuration in `database.yml`. See "Using Databases with JRuby" in Chapter 5 for more information.

Configuring Warbler

You will most likely need to edit the project's Warbler configuration, such as to list gems and libraries to include or to change the number of Rails instances. To do this you first need to create the configuration file (`Configuration/warble.rb`).

1. Right-click the project's node and choose Generate.
2. Select Warble from the Generator drop-down list, and click OK.
3. You can find the `warble.rb` file under the Configuration node.

You can also create the configuration file from a terminal window. Change to the project's folder and type the following command:

```
jruby -S warble config
```

The file has lots of comments to show how to configure the WAR file. You can learn more about configurations at `http://warbler.kenai.com`. Don't forget to add the JDBC adapter gem, if you are using it, as shown in this example:

```
 config.gems += ["activerecord-jdbcpostgresql-adapter"]
```

Creating the WAR File

Before you create the WAR file, check the `database.yml` file to make sure the production configuration is correct.

To create the WAR file, simply right-click the project's node and choose Run/Debug Rake Task. Then locate and double-click the `war` task. The task puts the WAR file in the project's top folder. If you don't see it from the Projects window, look in the Files window.

Deploying to Application Servers

Once you have a WAR file you are ready to deploy to a Java servlet container. Next we show how to deploy to the GlassFish application server and Tomcat.

Deploying to the GlassFish Application Server

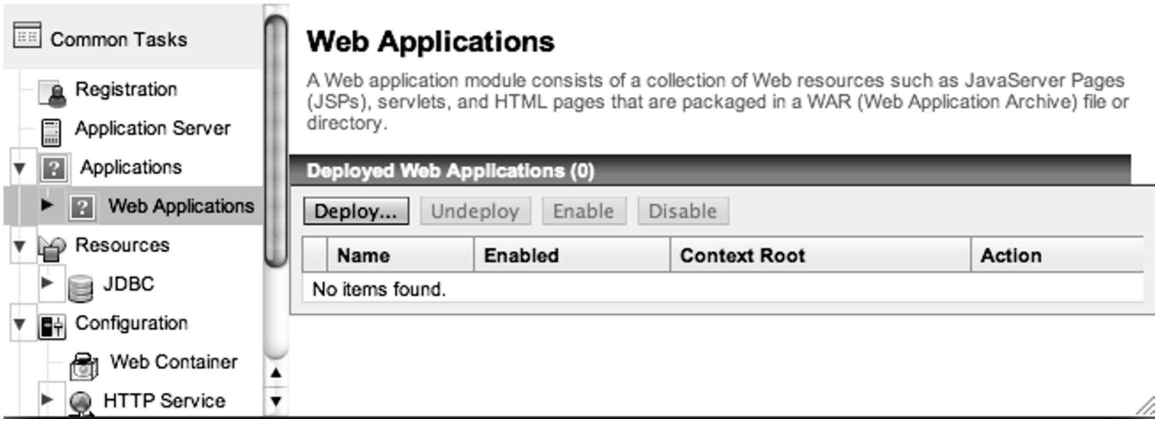
If you have been running your application from the IDE with the project's platform set to GlassFish V3 and that is where you want to deploy the WAR file, you might need to first undeploy the application, as these steps describe:

1. Go to the Servers window.
2. Expand Servers, GlassFish V3, and Applications.
3. Right-click the deployed application and choose Undeploy.

To deploy the WAR file from the IDE, follow these steps:

1. In the Services window, expand Servers.
2. If the server has not been started, right-click the node for the GlassFish server and choose Start.
3. Right-click the GlassFish node again and choose View Admin Console.
4. You might get a page that says `Your Application Server is now running`. When this happens, look for `To manage the server from localhost, click here`. Click the "here" link. Select the appropriate Internet connection and click OK. The browser installs and then opens the Administration Console.
5. If a login page appears, type **anonymous** for the login name. Leave the password blank.
6. In the Common Tasks window, select Applications ➤ Web Applications.
7. In the Web Applications window click Deploy (see Figure 9-3).

Figure 9-3. Web Applications Window in the Admin Server Console



Web Applications

A Web application module consists of a collection of Web resources such as JavaServer Pages (JSPs), servlets, and HTML pages that are packaged in a WAR (Web Application Archive) file or directory.

	Name	Enabled	Context Root	Action
No items found.				

8. Select the radio button for Local Packaged File or Directory That is Accessible from the Application Server.
9. Click Browse Files and then browse to and select the WAR file.

10. Click OK.
11. If you want, you can click Launch to run the application. If the page is cached, you might need to refresh the page to see the newly added entry.

If you want to deploy from the terminal window, you can use a command like this:

```
asadmin deploy schedules.war
```

To redeploy, add `-force=true`, as shown here:

```
asadmin deploy -force=true schedules.war
```

Deploying to Tomcat

You can deploy to Tomcat using these steps:

1. To autodeploy the application, copy the WAR file to the `webapps` folder under `TOMCAT_HOME`. If Tomcat is registered with the IDE, you can find the value of `TOMCAT_HOME` by right-clicking the Apache Tomcat node in the Servers window and choosing Properties. Look at the value in the Catalina Base field.
2. In the browser, enter the URL **http://localhost:8080/application-name**. If Tomcat is registered with the IDE you can also expand the Apache Tomcat node in the Servers window, expand Web Applications, right-click the application's node, and choose Open in Browser. If you do not see the application's node, right-click the Apache Tomcat node and choose Restart.

Summary

The Warbler gem and its associated plugin make it possible to package and deploy your Rails application to a Java servlet container. Once the Warbler plugin is installed, NetBeans supports generating the Warbler configuration file and running its Rake tasks to create the deployable WAR file. The WAR file can be deployed to any Java servlet container, and we concluded with examples on how to deploy to GlassFish and Tomcat.

Chapter 10: Customizing the IDE

Now that you are familiar with using the IDE for Ruby development, you probably want to tweak the environment to fit the way you are used to working. For example, you might want to change the shortcuts to match your previous editor, or perhaps your company has coding standards that you would like to work into the templates. As you will learn in this chapter, the NetBeans IDE offers lots of options for configuring the IDE so that it works best for you.

Working with Editor Options

The Editor panel in the Options dialog box allows you to modify several editing options. To open the panel, select Tools ➤ Options (NetBeans ➤ Preferences on the Mac) from the main toolbar. Then click Editor to display the Editor panel. Alternatively, type **editor** in the Quick Search text box and choose Editor Options from the Options category.

Note See the "Tweaking Under the Hood" section later in this chapter for more editor options.

Tuning the Code Completion Pop-Up

The General tab in the Editor panel provides several code completion options (see Figure 10-1):

Turning off the autocompletion pop-up. When you type a period (`.`) or a double-colon (`::`), the IDE automatically opens the code completion pop-up. To turn this off, clear the Auto Popup Completion Window check box. You might also want to clear the Auto Popup Documentation check box. After you turn off autocompletion, you can still display the code completion pop-up by pressing Ctrl+Space (Cmd+Space on the Mac). To display documentation, press Ctrl+Shift+Space (Cmd+Shift+Space on the Mac).

Turning off autocompletion of single choices. If there is only one possible completion when you press Ctrl+Space, the IDE automatically completes the code. For example, if you type **"hello".bytesi** and press Ctrl+Space, the IDE completes the code as `"hello".bytesize`. If you prefer that the IDE display the pop-up instead, clear the Insert Single Proposals Automatically check box.

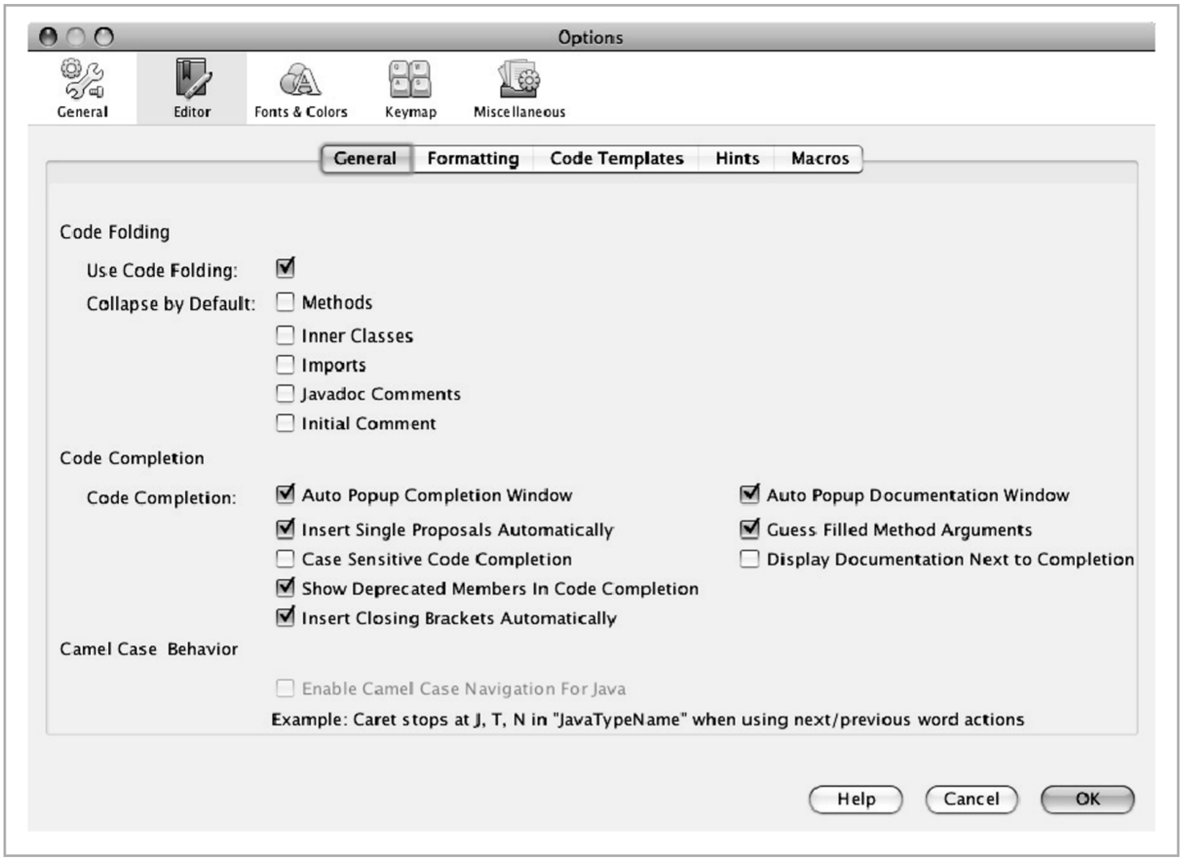
Making code completion case sensitive. If you type **Action_N** and press Ctrl+Space, the code completion pop-up suggests `action_name`. If you want the pop-up to be case sensitive, select the Case Sensitive Code Completion check box.

Turning off the automatic insertion of closing end, bracket, parenthesis, brace, or quotation mark. If the automatic insertion of ending delimiters interferes with your workflow, you can turn it of by clearing the Insert Closing Brackets Automatically check box.

Displaying the documentation below or to the side of the pop-up. By default, the documentation appears under the code completion pop-up. If you prefer that it appear to the side of the pop-up, select the Display Documentation Next to Completion check box.

Note The Show Deprecated Members in the Code Completion check box and the Guess Filled Method Arguments check box do not apply to Ruby or Rails projects.

Figure 10-1. Code Completion Options



Code Folding

Use Code Folding: ✓

Collapse by Default:
- ☐ Methods
- ☐ Inner Classes
- ☐ Imports
- ☐ Javadoc Comments
- ☐ Initial Comment

Code Completion

Code Completion:
- ✓ Auto Popup Completion Window
- ✓ Insert Single Proposals Automatically
- ☐ Case Sensitive Code Completion
- ✓ Show Deprecated Members In Code Completion
- ✓ Insert Closing Brackets Automatically
- ✓ Auto Popup Documentation Window
- ✓ Guess Filled Method Arguments
- ☐ Display Documentation Next to Completion

Camel Case Behavior

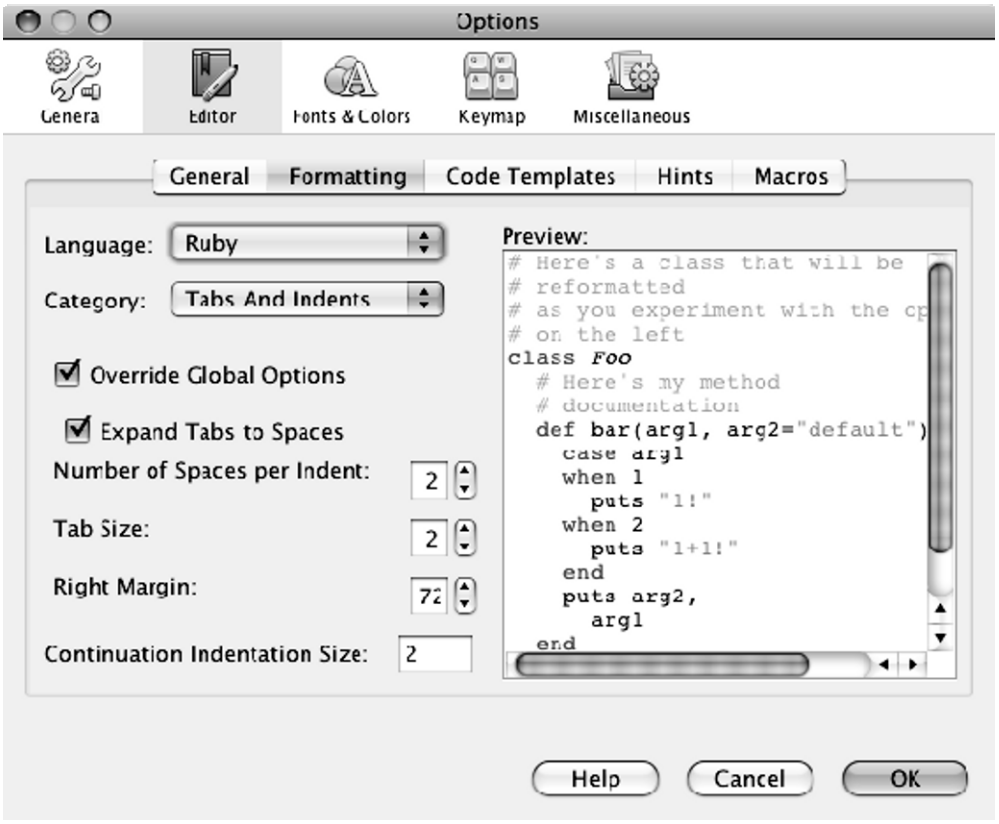
- ☐ Enable Camel Case Navigation For Java

Example: Caret stops at J, T, N in "JavaTypeName" when using next/previous word actions

Setting Formatting Options

To modify the tab, indentation, margin, and comment reformatting options, click the Formatting tab on the Editor panel and then choose Ruby from the Language drop-down list (see Figure 10-2). Alternatively, type **formatting** in the Quick Search text box. These properties are pretty self-explanatory. The Preview panel shows how your changes affect the code.

Figure 10-2. Formatting Options



Language: Ruby

Category: Tabs And Indents

☑ Override Global Options

☑ Expand Tabs to Spaces

Number of Spaces per Indent: 2

Tab Size: 2

Right Margin: 72

Continuation Indentation Size: 2

Preview:

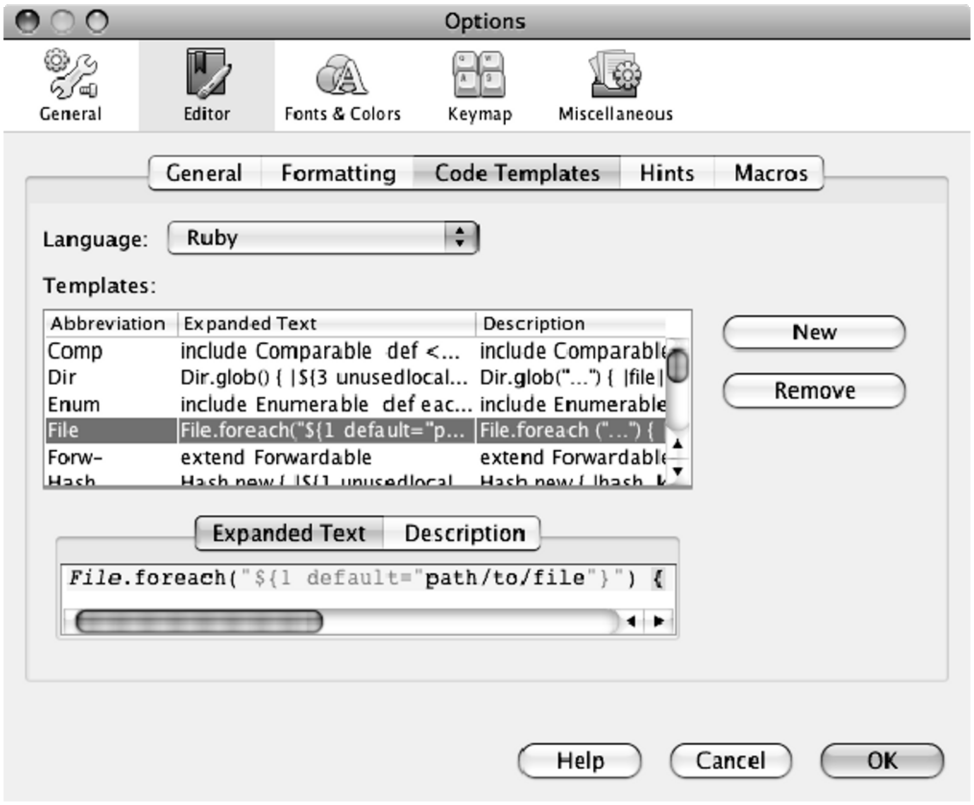
```
# Here's a class that will be
# reformatted
# as you experiment with the op
# on the left
class Foo
  # Here's my method
  # documentation
  def bar(arg1, arg2="default")
    case arg1
    when 1
      puts "1!"
    when 2
      puts "1+1!"
    end
    puts arg2,
      arg1
  end
```

Tip You can also customize the formatting settings for a specific project. Right-click the project's node and choose Properties from the pop-up menu. In the Project Properties dialog box, select the Formatting category.

Customizing Live Code Templates

In Chapter 6, you learned about live code templates, where you can type a few characters and press Tab to expand the characters into a snippet of code. If you want to alter or add templates, go to the Code Templates tab in the Editor panel. Next, choose a language type from the Language drop-down list. Figure 10-3 shows the live code templates for the Ruby language.

Figure 10-3. Ruby Live Code Templates



Language: Ruby

Templates:

Abbreviation	Expanded Text	Description
Comp	include Comparable def <...	include Comparable
Dir	Dir.glob() { |${3 unusedlocal...	Dir.glob("...") { |file|
Enum	include Enumerable def eac...	include Enumerable
File	File.foreach("${1 default="p...	File.foreach ("...") {
Forw-	extend Forwardable	extend Forwardable
Hash	Hash.new { |${1 unusedlocal	Hash.new { |hash, k

Expanded Text | Description

```
File.foreach("${1 default="path/to/file"}") {
```

Tip You can access the Code Templates tab by typing **template** in the Quick Search text box.

To modify a template, select it from the Templates list and use the Expanded Text tab to change the text. If you want to add a template, click New, type an abbreviation, and click OK. Then provide the expanded text. Table 10-1 lists the parameters you can use in the expanded text. You can use the Description tab to write an explanation if you want.

Table 10-1. Expanded Text Parameters

PARAMETER	EXPLANATION
`${`*`parameter-name`*` [default="`*`value`*`"] [editable="false"] }`	Input parameter. You can optionally specify a default value or disable the editing of the parameter.
`${cursor}`	Where to put the caret after expansion.
`${selection}`	Where to paste the contents of the editor selection.
`${class}`	Name of the class that the template location is in.
`${classfqn}`	Fully-qualified name of the class that the template location is in.
`${superclass}`	Fully-qualified name of the superclass of the class that the template location is in.
`${method}`	Name of the method that the template location is in.

Table 10-1. Expanded Text Parameters (continued)

PARAMETER	EXPLANATION
`${methodfqn}`	Fully-qualified name of the method that the template location is in.
`${file}`	Name of the current file.
`${path}`	Name of the current full file path.
`${var unusedlocal defaults="i,j,k"}`	Picks an unused local variable in the current scope; it tries each default until it finds an unused one. If all are used, it creates a unique name.

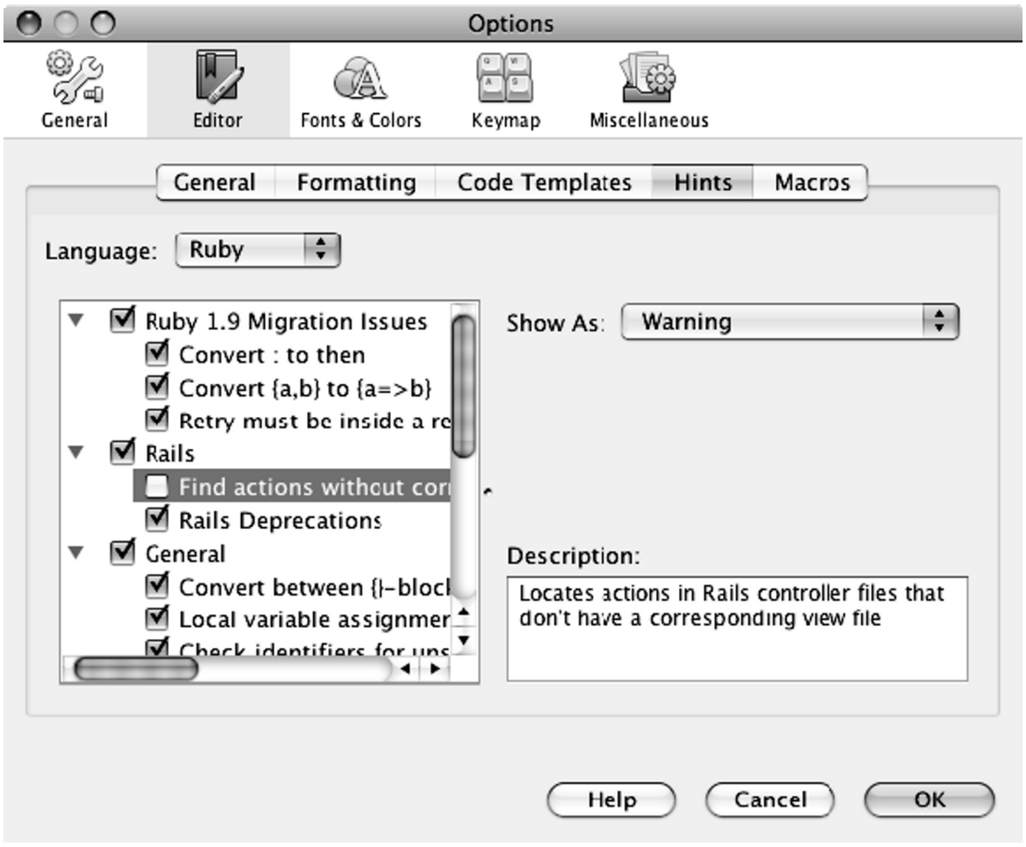
To learn more about creating and editing live code templates, go to `http://wiki.netbeans.org/RubyCodeTemplates`.

Tip The Expand Template On drop-down list offers different key combinations for expanding the characters into a template. For example, you can have the characters expand into the template when you press Shift-Space, instead of when you press Tab.

Adjusting Hints

To customize which hints the editor displays, go to the Hints tab (see Figure 10-4). To quickly open this tab, type **hints** in the Quick Search text box. You can select or clear the check boxes to customize which hints the editor displays. You can also specify which hints should display as warnings, which should display as errors, and which should not be displayed unless your caret is on the specific line.

Figure 10-4. Hint Tab



Language: Ruby

- ☑ Ruby 1.9 Migration Issues
  - ☑ Convert : to then
  - ☑ Convert {a,b} to {a=>b}
  - ☑ Retry must be inside a re
- ☑ Rails
  - ☐ Find actions without cor
  - ☑ Rails Deprecations
- ☑ General
  - ☑ Convert between {|-block
  - ☑ Local variable assignmer
  - ☑ Check identifiers for unc

Show As: Warning

Description:

Locates actions in Rails controller files that don't have a corresponding view file

Creating Code Shortcuts

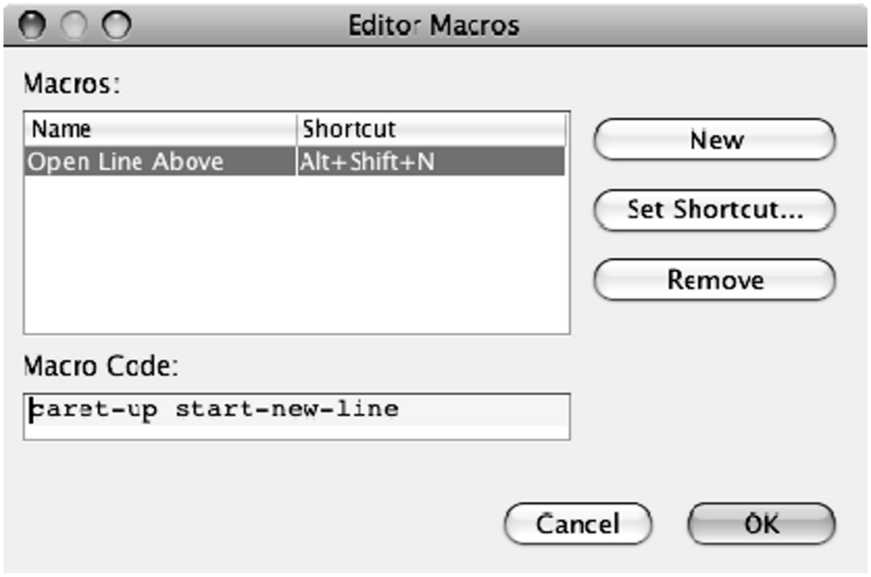
You can create shortcuts for common editing actions. You can use the Macros tab in the Editor panel to create the shortcut. However, it is easier to create a shortcut by recording your editing keystrokes, as shown in the following steps:

1. To start recording, choose Edit ➤ Start Macro Recording from the main menu.
2. In the editor, complete the steps that you want to record.

3. Choose Edit ➤ Stop Macro Recording.
4. Type a Macro Name and click OK. The macro appears in the Editor Macros dialog box (see Figure 10-5).
5. Click Set Shortcut.
6. Type the shortcut. For example to set the shortcut to Alt+Shift+N, press the Alt key and hold it down. Next press the Shift key, and then press the N key.
7. Click OK.

Tip You can use cut-and-paste actions to create a macro that wraps selected text with characters, such as `<%#` and `%>`. Select some text and start the macro recording. Press Ctrl+X, type the leading characters, press Ctrl+V, and type the ending characters.

Figure 10-5. Editor Macros Dialog Box

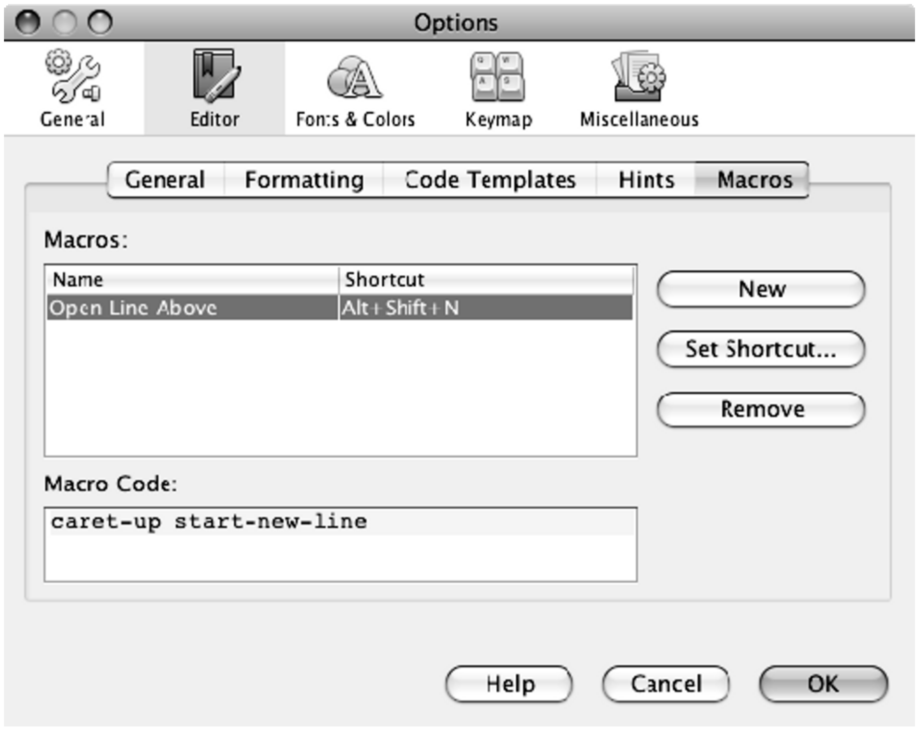


Name	Shortcut
Open Line Above	Alt+Shift+N

```
caret-up start-new-line
```

After you create your macro you can use the Macros tab in the Editor panel to edit the macro, remove it, or change its shortcut (see Figure 10-6). To quickly access this tab, type **macro** in the Quick Search text box.

Figure 10-6. Editor Macros Tab



Macros:

Name	Shortcut
Open Line Above	Alt+Shift+N

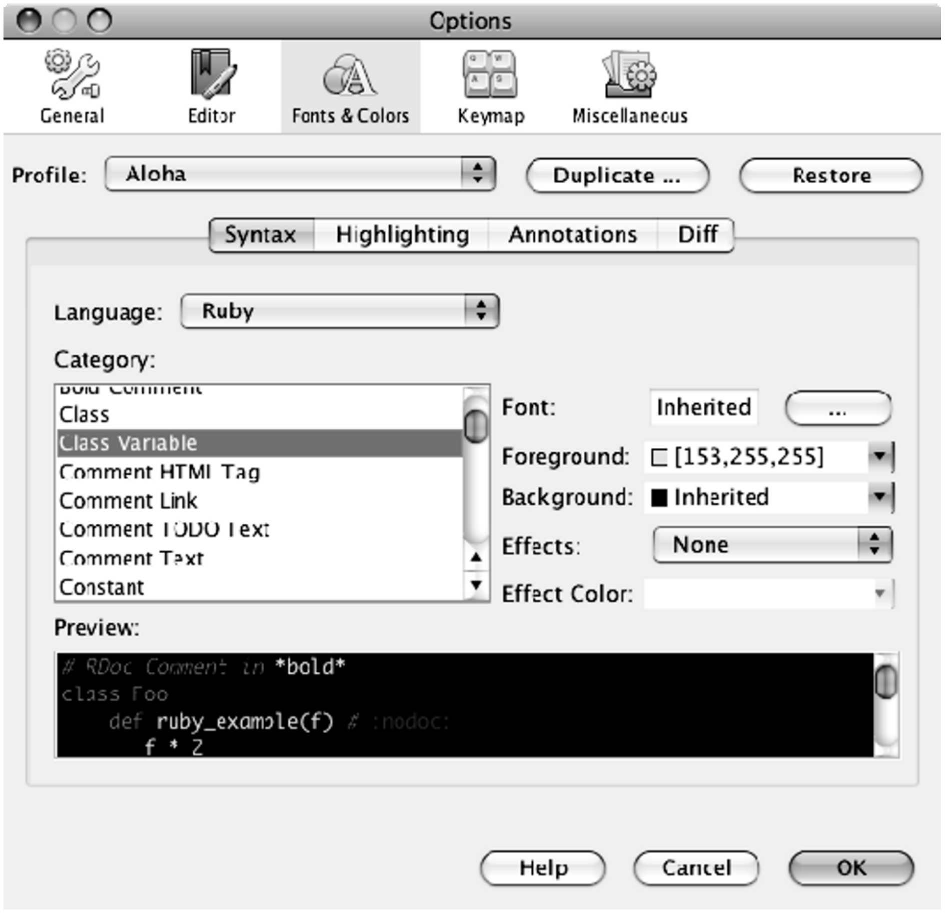
Macro Code:

```
caret-up start-new-line
```

Changing Fonts and Colors

In Chapter 6 we talked about semantic styles. As you would expect, you can change these styles if you are more familiar with a different editor. The Fonts and Colors panel in the Options dialog box lets you specify the font, foreground color, background color, and effect (underlined, wave underlined, or strikethrough) for each category of a specific Language type, such as Ruby or Embedded HTML (see Figure 10-7). You can quickly access this tab by typing **fonts** in the Quick Search text box.

Figure 10-7. Fonts and Colors Panel



Profile: Aloha

Language: Ruby

Category:
Bold Comment
Class
Class Variable
Comment HTML Tag
Comment Link
Comment TODO Text
Comment Text
Constant

Font: Inherited
Foreground: □ [153,255,255]
Background: ■ Inherited
Effects: None
Effect Color:

Preview:

```
# RDoc Comment in *bold*
class Foo
    def ruby_example(f) # :nodoc:
       f * 2
```

The IDE offers a few different profiles in addition to the standard NetBeans profile, such as Norway Today and City Lights. You can try them out and easily switch back to the standard profile if you don't like them. Or you can duplicate a profile and modify it as you wish. There is a problem with the City Lights profile because the background is black and the foreground color for class names, superclasses, and constants is black as well. You can change the foreground or background color for all of these by selecting

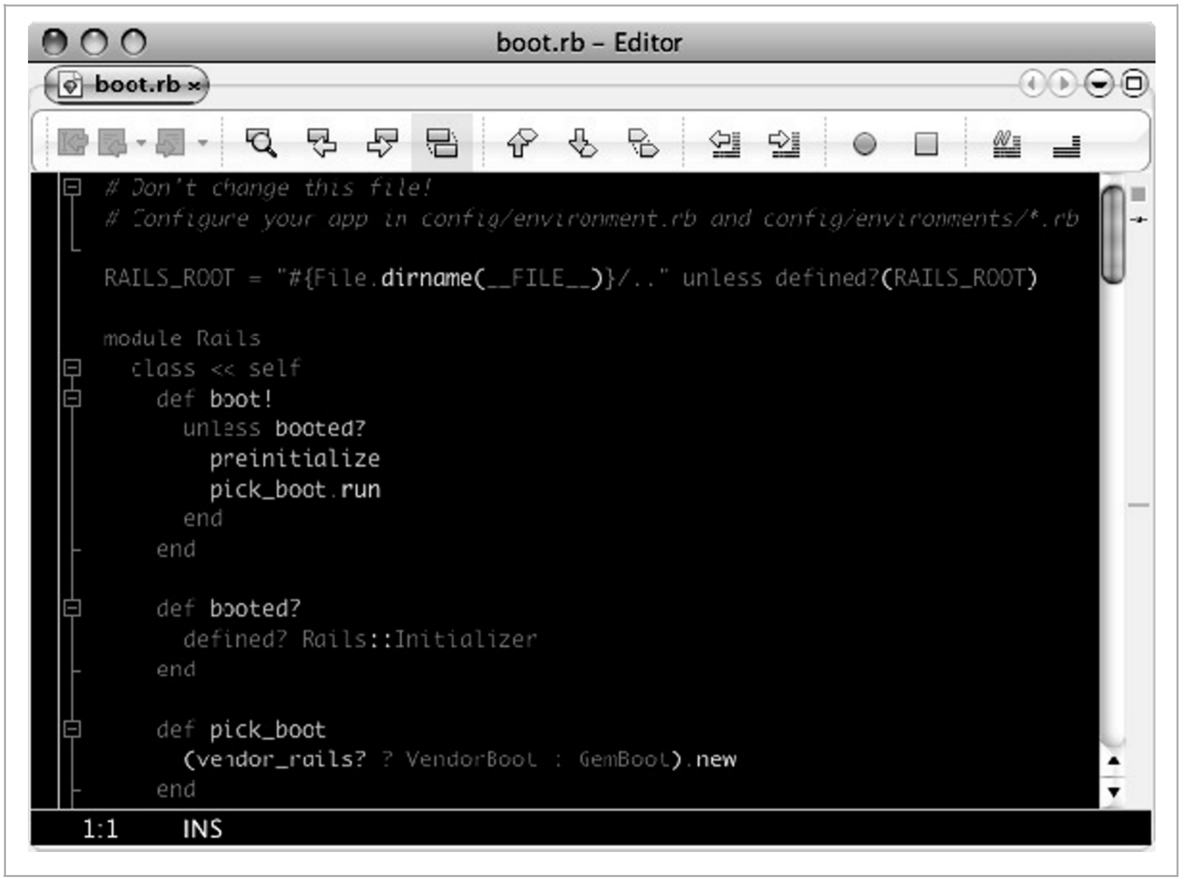
Ruby from the Language drop-down list and editing the Constant category. You also need to edit the Global Variable category.

You can get Ruby-specific themes, such as Aloha (see Figure 10-8) and Ruby Dark Pastels, from the update center by following these steps:

1. Choose Tools ➤ Plugins.
2. Scroll to the Ruby category and select Extra Ruby Color Themes.
3. Click Install.
4. Complete the installer steps and restart the IDE.
5. Open the Options dialog box and click Fonts and Colors.
6. Select All Languages from the Language drop-down list or select a specific Language type.
7. Choose the desired theme from the Profile drop-down list.

Note By the time you read this, these profiles might be part of the product. There also might be other profiles in the plugin.

Figure 10-8. Aloha Theme in the Editor

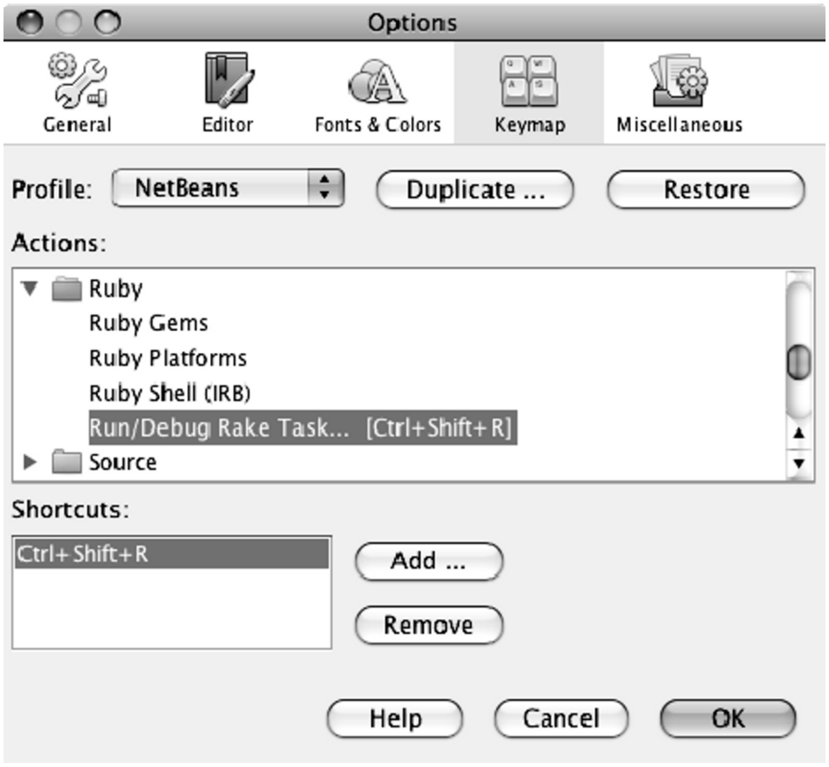


```
# Don't change this file!
# Configure your app in config/environment.rb and config/environments/*.rb

RAILS_ROOT = "#{File.dirname(__FILE__)}/.." unless defined?(RAILS_ROOT)

module Rails
  class << self
    def boot!
      unless booted?
        preinitialize
        pick_boot.run
      end
    end

    def booted?
      defined? Rails::Initializer
    end

    def pick_boot
      (vendor_rails? ? VendorBoot : GemBoot).new
    end
```

Modifying Keyboard Shortcuts

If you are used to a different editor's keyboard shortcuts, you might prefer to change NetBeans' default shortcuts instead of learning a new set. The IDE provides shortcut settings for Eclipse and Emacs. You can switch to those profiles or create your own. You can also change individual keymap settings. To edit the shortcuts, open the Options dialog box and click Keymap (see Figure 10-9). To quickly access this tab, type **shortcuts** in the Quick Search text box.

Figure 10-9. Editor Keymap Panel



Profile: NetBeans

Actions:

- Ruby
  - Ruby Gems
  - Ruby Platforms
  - Ruby Shell (IRB)
  - Run/Debug Rake Task... [Ctrl+Shift+R]
- Source

Shortcuts:

Ctrl+Shift+R

Select Eclipse or Emacs from the Profile to switch to that set of shortcuts. If you want to alter a profile's settings, you should click Duplicate to create your own set and modify those settings.

To create a new shortcut, select the action. Then click Add and press the key sequence. For example, press Alt and hold it down, then press N for the key combination Alt+N. If the key combination is already used for a different action, the IDE displays a message.

After you change to a different profile or change individual shortcuts, you can create a new shortcut cheat sheet. Because there is no menu action or shortcut to perform this task, you must use the Quick Search feature. Type **export** in the Quick Search text box and select Export Shortcuts to HTML.

The drop-down list might display `(no results)`, but it still creates the cheat sheet. You can find the cheat sheet under `config` in your NetBeans user directory, for example *home*`/.netbeans/6.5/config/shortcuts.html`. The cheat sheet uses the abbreviations shown in Table 10.2.

Table 10.2. Cheat Sheet Abbreviations

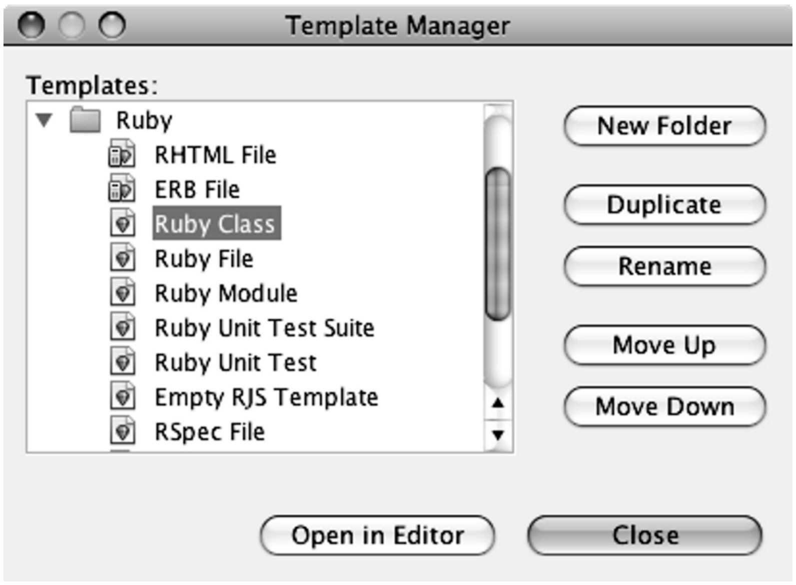
ABBREVIATION	EXPLANATION
D	Ctrl key (Cmd key on the Mac)
O	Alt key (Ctrl key on the Mac)
S	Shift key

Tip You can view the shortcut cheat sheet from the IDE by choosing File ➤ Open File from the main menu. Right-click in the open file and choose View to open the file in a browser. Alternatively, add *home*`/.netbeans` to the Favorites window so you have quick access to the cheat sheet as well as the other files in your NetBeans user folder.

Modifying File Templates

Whenever you use the New File editor, the IDE uses a file template to build the file. If you have certain standards or conventions that you want the templates to follow, you can use the Template Manager to alter the template's contents (see Figure 10-10).

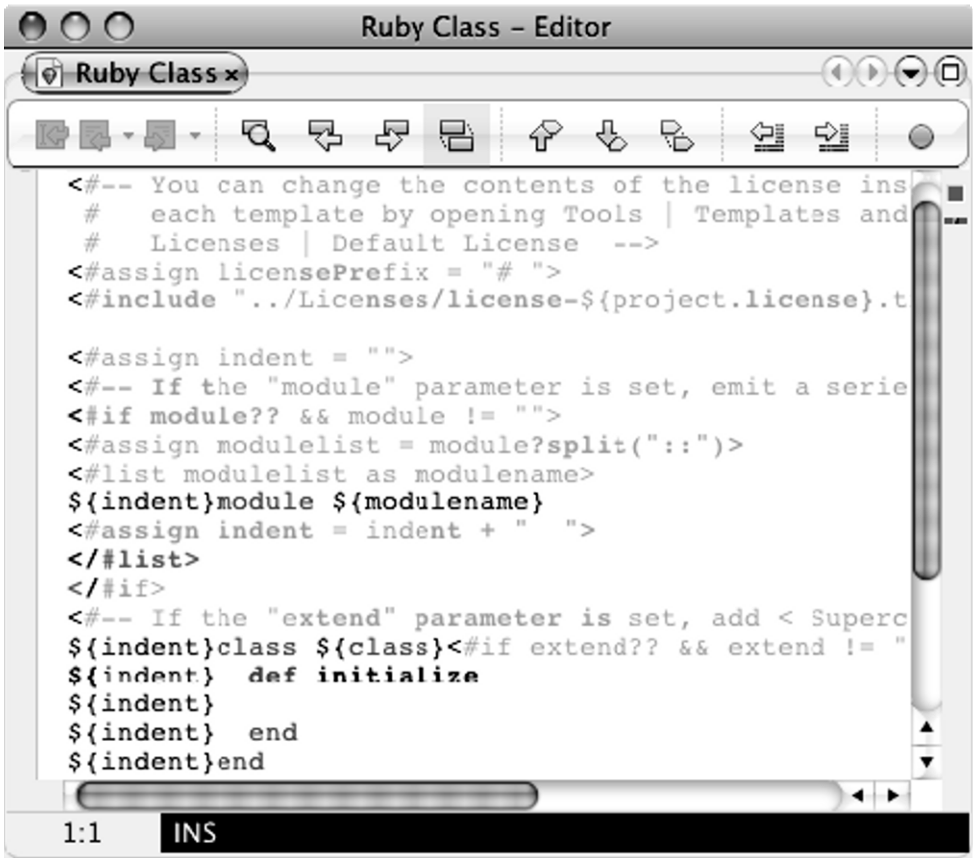
Figure 10-10. Template Manager



Editing a Template

To edit a template choose Tools ➤ Templates from the main menu or type **template** in the Quick Search text box and press Enter. In the Template Manager, expand Ruby, select the template, and click Open in Editor. Figure 10-11 shows the Ruby Class template in the editor.

Figure 10-11. Ruby Class Template



```
<#-- You can change the contents of the license ins
 #   each template by opening Tools | Templates and
 #   Licenses | Default License  -->
<#assign licensePrefix = "# ">
<#include "../Licenses/license-${project.license}.t

<#assign indent = "">
<#-- If the "module" parameter is set, emit a serie
<#if module?? && module != "">
<#assign modulelist = module?split("::")>
<#list modulelist as modulename>
${indent}module ${modulename}
<#assign indent = indent + "  ">
</#list>
</#if>
<#-- If the "extend" parameter is set, add < Superc
${indent}class ${class}<#if extend?? && extend != "
${indent}  def initialize
${indent}
${indent}  end
${indent}end
```

The IDE uses the FreeMarker template engine, which enables you to use a combination of text, comments, and the FreeMarker Template Language (FTL) tags to design your template. To learn about FTL markup, go to `http://freemarker.org/docs/index.html`.

Using Template Properties

Table 10-3 shows the predefined properties you can use in a template.

Table 10-3. Predefined Template Properties

PROPERTY	**EXPLANATION**
`${date}`	Current date, in the format MMM DD, YYYY
`${encoding}`	Default encoding, such as UTF-8
`${name}`	Name of the file
`${nameAndExt}`	Name of the file including its extension
`${time}`	Current time, in the format H:MM:SS PM
`${user}`	User name

When you view the Ruby file templates, you will notice some properties that are specific to Ruby and Rails projects or specific to that file type, such as `${indent}`, `${modulename}`, and `${extend}`.

To create new properties or override the values of predefined properties, follow these steps:

1. Expand the User Configuration Properties node in the Templates list and select User.properties.
2. Click Open in Editor.
3. Use the following key/value syntax to set a property value: *property=value*.

Listing 10-1 shows an example of adding and overriding properties.

Listing 10-1. Contents of User.properties

```
# Overrides current value
user=Alice Timmons
# new properties
userEmail=alice@land.com
copyright=2008 Wonderful Ruby Industries, Inc.
```

If you add the code in Listing 10-2 to the Ruby Class template, new Ruby Class files will be created with the comments shown in Listing 10-3.

Listing 10-2. Comments Added to the Ruby Class Template

```
# TODO Put class definition here
#
# Author:: ${user} (mailto:${userEmail})
# Copyright:: (c) ${copyright}
```

Listing 10-3. Output in the New Ruby Class

```
# To change this template, choose Tools | Templates
# and open the template in the editor.

# TODO Put class definition here
#
# Author:: Alice Timmons (mailto:alice@land.com)
# Copyright:: (c) 2008 Wonderful Industries, Inc.
class MyClass
  def initialize

  end
end
```

Adding Licenses to a Template

There are two ways to include a license in a new file: you can modify the default license or you can add your own licenses and specify which license to use for each file type.

Modifying the Default License

When a license is not specified the IDE uses the default license. You can change the wording in this license as follows:

1. Expand Licenses, select Default License, and click Open in Editor.
2. Modify the file to include your license text. All new files will include this license.

Adding Your Own Licenses

You can create license templates and then add user properties to reference the license templates, as shown here:

1. Expand Licenses, select Default License, and click Duplicate.
2. Click Rename and name the file `license-`*`license-name`*`.text`. For example, name it `license-ruby.txt`.
3. Change the argument in the template's license include statement to reference a user property instead of a project property, as shown in the following line:

```
<#include "../Licenses/${rubyLicense}.txt">
```

4. Next, add a license property to the User.properties template, as shown in the following example:

```
 rubyLicense=ruby
```

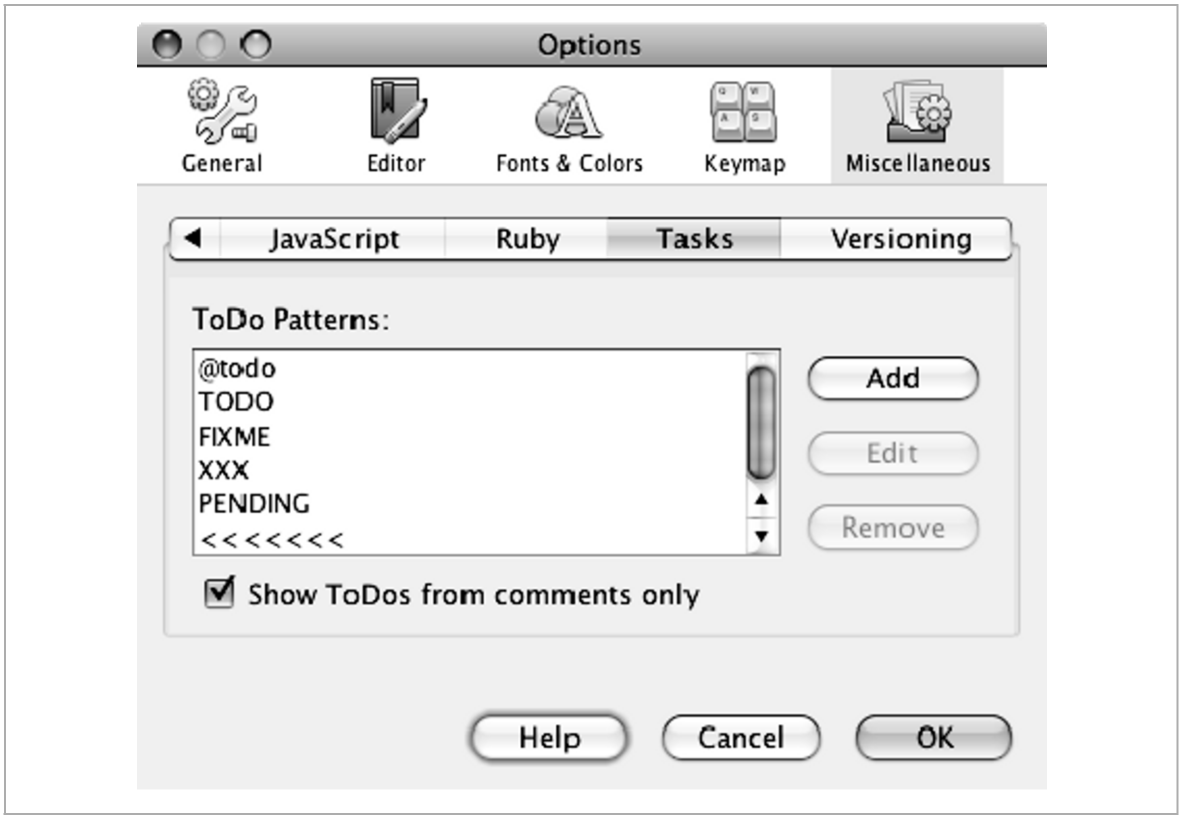
Note In a future release, you might be able to set the license on a project-by-project basis by adding `project.license=`*`license-name`* to a project's `nbproject/project.properties` file.

Setting Task List Patterns

In Chapter 6 we showed how to use the task list feature to quickly view notes to yourself about changes or fixes you need to make. The IDE lists all the comments that contain the text `TODO`, `@todo`, `FIXME`, `XXX`, `PENDING`, or `<<<<<<<`. If you would like to add more patterns, open the Tasks tab in the Miscellaneous pane in the Options dialog box (see Figure 10-12). You can change and remove existing patterns in addition to adding new ones. You can also clear the Show ToDos From Comments Only check box if you want to search for patterns in the code too.

Tip To quickly open the Tasks tab, type **TODO** in the Quick Search text box.

Figure 10-12. Tasks Tab



ToDo Patterns:

```
@todo
TODO
FIXME
XXX
PENDING
<<<<<<<
```

☑ Show ToDos from comments only

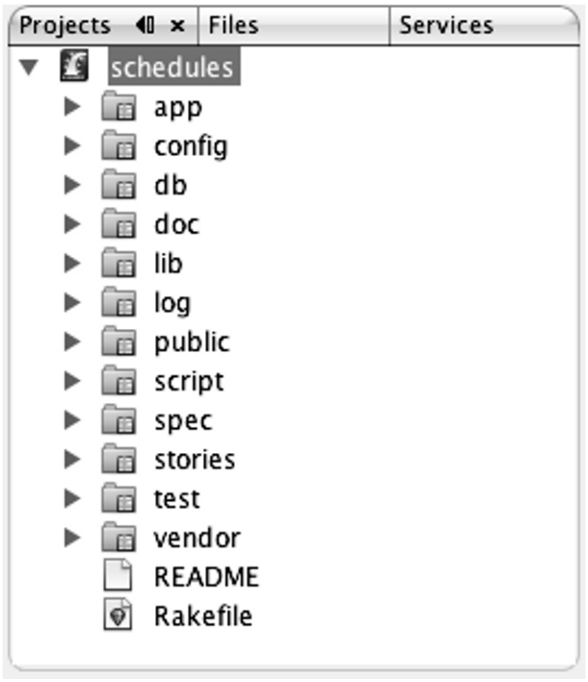
Working with Nodes, Files, and Folders

Here are some options for configuring how the IDE interacts with file system objects.

Displaying the Physical File Structure in the Projects Window

If you have been working with Ruby and Rails projects for a while and are accustomed to the Rails directory structure, you might prefer that the Projects window show the physical file structure instead of the logical one. You can easily switch to the physical view by opening the Options window, clicking Miscellaneous, and then clicking the Ruby tab. Clear the Show Logical Project View check box and restart the IDE. Figure 10-13 shows the Projects window after the check box is cleared. Just as before, you can right-click on a node to get pop-up menus with development-specific actions. The pop-up menus in the Files window still offer the file system actions.

Figure 10-13. Physical Structure in the Projects Window



Adding Ruby Script File Extensions

Follow these steps to enable the editor to recognize other file extensions, such as `.red` or `.sty`, as ruby script files:

1. Open the Files tab in the Miscellaneous pane in the Options dialog box.
2. Click New.
3. Type the extension, such as **red**, and click OK.
4. Select Ruby Files (`text/x-ruby`) from the Associated File Type (MIME) drop-down list.
5. Click OK.

Changing the Location of the Ruby Source Files Folder

By default, the IDE puts the source files under the `lib` folder. When you create a project from existing sources you can specify the source folder's location. If you want to change the location of source files for a project you start from scratch, follow these steps:

1. Right-click the project's node and choose Properties.
2. Delete the entry for the `lib` folder or change its label. To change the label, click in the Label column and type a new name.
3. Click Add Folder and navigate to the desired folder (you can click the New Folder icon in the Add Source Folder dialog box to create the folder if it doesn't exist).
4. Click Open.
5. Change the entry's label to Source Files.
6. From now on, when you add a Ruby class, it will go in the new folder by default. When you run the project, it looks for the main file in the new folder.

Editing Hidden Files in the IDE

By default, the IDE does not display hidden files (files with names that begin with a period) in either the Projects or Files window. To display these files, open the Options dialog box, click Miscellaneous, and click the Files tab. Cut the string from the Ignored Files Pattern text box and click OK. The hidden files now appear. You can replace the string or click Default to restore to the NetBeans default.

Tip You can also access a hidden file by choosing File ➤ Open File from the main menu, then navigating to the file, selecting it, and choosing Open.

Tweaking Under the Hood

Some IDE options, such as the IDE's font size, need to be set at start-up time. In addition, the engineers often add options for enabling experimental features. These features usually get incorporated in a later release, but are not currently available from the user interface. (Some of these experimental features might already be introduced by the time you read this.)

In this section we talk about the options you might want to use in your Ruby and Rails development. To enable the features you must pass the option to the IDE upon startup. There are two ways to do this:

1. Set the option in the configuration file, which can be found at `netbeans-install-dir/etc/netbeans.conf`. Add the option to the space-separated list for the `netbeans_default_options` switch. You can also create an `etc` folder in your NetBeans user directory and put a personal copy of `netbeans.conf` in that folder. This is a good way to keep your configurations when you upgrade the IDE. Your personal configurations override the global configurations.

2. Pass the option to the IDE at the command line. For example when you start the IDE with the following command, the comment continuation option is passed to the IDE.

```
netbeans -J-Druby.cont.comment=true
```

IDE Options

The following options enable you to configure the IDE look and feel or behavior in ways that you might find useful.

Change the JDK That the IDE Runs On

If you upgrade the JDK after you install the IDE, you can make the IDE use the new JDK by passing the following option in the startup command:

```
--jdkhome path-to-jdk
```

Alternatively, edit (or add) the following line in the `netbeans.conf` file:

```
netbeans_jdkhome="path-to-jdk"
```

Suppressing Browser Window Startup

Every time you click the Run or Debug button, the IDE opens up a browser window and navigates to the root page. Next thing you know you have a bunch of browser windows cluttering your desktop. You might find it advantageous to use the following option to turn off browser startup. Then, when you start your work session, open up a browser and type in the URL for the application. Just keep this browser window open and type in the URLs as needed. Clicking the Run or Debug button starts a new session; however, you won’t get a new browser window.

```
-J-Drails.nobrowser=true
```

Opening a File in NetBeans from the Command Line

The `--open` option enables you to start up the IDE with the specified file opened in the editor. You can also use this option from the command line to open a file in a running IDE (assuming `netbeans` is in your path).

```
 netbeans –open MyRubyClass.rb
```

Setting Window, Menu, and Tooltip Font Sizes

Use the `--fontsize` *`value`* option to change the size of the fonts that are used in the windows and menus. When demonstrating the software on a projection screen, we like to use `--fontsize 18`. At that size the characters have a nice thickness.

Changing the Look and Feel

You can easily switch to the Metal look and feel by using the following option:

```
 --laf javax.swing.plaf.metal.MetalLookAndFeel
```

If you have installed the JDK update 10 software and `netbeans_jdkhome` points to this installation, you can use the new Nimbus look and feel with this option:

```
 --laf Nimbus
```

For more look-and-feel options, go to `http://wiki.netbeans.org/FaqCustomLaf`.

Turning on Detailed Ruby Logging

Use the following options to generate more detailed Ruby logging. When you use these options, the IDE writes all the Ruby and Rails commands that it sends to the command line to the IDE's log. If you are having problems, you can try out the same commands in a terminal window to get further feedback. To see the log file's contents, choose View ➤ IDE Log from the main menu.

```
-J-Dorg.netbeans.modules.ruby.level=400
-J-Dorg.netbeans.api.ruby.platform.level=400
```

Note The parameter values might change in a patch or future release. You should consult `http://wiki.netbeans.org/FaqRubyNBLogging` to verify which values to use.

If you are diagnosing debugger problems, use the following options to generate detailed Ruby debugger logging:

```
-J-Dorg.rubyforge.debugcommons.level=300
-J-Dorg.rubyforge.debugcommons.verbose=true
```

For more information see `http://wiki.netbeans.org/RubyDebugging#section-RubyDebugging-HowToFileABug`.

Editor Options

The Ruby engineers have created the following options to give you access to some experimental features. You can get details about these options and more at `http://wiki.netbeans.org/RubyOptions`.

Defining Word Boundaries

Many of the editor shortcuts, such as Ctrl+Delete (Cmd+Delete on the Mac), work within word boundaries. By default, the editor considers an embedded capital letter as a word boundary. For example, if the cursor is in front of `ActiveRecord`, pressing Ctrl+Delete deletes `Active` but not `Record`. The same is true for underscores. If the cursor is in front of `const_missing`, Ctrl+Delete removes `const` and leaves `_missing`. If you prefer that the IDE not use embedded capitals and underscores as word boundaries, use the following option to turn this off:

```
-J-Dno-ruby-camel-case-style-navigation=true
```

Automatically Wrapping Comments

If you would like the IDE to automatically wrap your comments as you type, and place the comment character at the beginning of each new line it adds, use the following option:

```
-J-Druby.autowrap.comments=true
```

Automatically Continuing Comment Lines

When you press Enter at the end of a comment line, the IDE adds a blank line. If you prefer that it add a line that begins with a comment character, use the following option:

```
-J-Druby.cont.comment=true
```

Excluding Parentheses in Code Completion

Because parentheses are optional, code completion looks at a method's documentation to determine if it should surround the arguments with parentheses when it writes out the method call. If you prefer code completion to never use parentheses, use the following option:

```
-J-Druby.complete.spaces=true
```

Troubleshooting the IDE

If you are unable to start the IDE or you are having serious problems with the IDE, the cause might be errors in your settings. If you have such problems, we recommend that the first step you take is to start with a fresh user directory. You have three ways to do this:

- Rename your current NetBeans user directory.
- Edit the `netbeans_default_userdir` property in the `*netbeans-install-dir*/etc/netbeans.conf` file to point to a

different folder. For example you might change it to `netbeans_default_userdir="${HOME}/.netbeans/test"`.

- Start the IDE using the `--userdir` property, as shown here: `netbeans --userdir /tempuserdir`

Restart the IDE and see if it corrects the problem.

If restoring the user settings to the initial default values does not fix the problem, look at the IDE's log file by choosing View ➤ IDE Log File from the main menu. If you are not able to run the IDE, the log file is located in *`netbeans-user-dir`*`/var/log`. You might want to turn on detailed Ruby logging, which we talked about earlier in this chapter, to get more details in the log file.

There are several places to go for assistance. For general IDE issues you can go to `http://forums.netbeans.org`. There you will find the NetBeans Users forum, which is for general questions about the IDE, and the Ruby Users forum, which is for the Ruby-specific features. Another place that you might find helpful is `http://wiki.netbeans.org/RubyFAQ`. You can also peruse the NetBeans Issues archive or file a bug, feature request, or enhancement request by going to `http://netbeans.org/community/issues.html`.

When asking questions in the forums, it is a good idea to note which version of the IDE you are using, which interpreter (JRuby or Ruby) and which version, and what operating system you are on.

Summary

As you gain experience with the NetBeans IDE, you come to learn that it appears to be endlessly customizable. The NetBeans editor allows you to customize code folding, code completion, formatting, templates, hints, fonts, colors, and task list patterns, as well as record your favorite actions as macros. The existing keyboard shortcuts are all configurable, and you can easily define new ones to match your favorite actions.

The templates that define each new file are also easily accessible and configurable, including the license that appears in your code.

In addition, it is possible to change how project files are displayed, which project files are displayed, and their default locations. Finally, this chapter concludes with a sampling of some of the "under the hood" options you can use to really fine tune and understand how the NetBeans IDE is working for you.