

# Ruby in STEEL

Ruby On Rails Development  
in Visual Studio 

USER GUIDE / MANUAL TO RUBY IN STEEL

SAPPHIRESTEEL SOFTWARE  
[HTTP://WWW.SAPPHIRESTEEL.COM](http://www.sapphiresteel.com)

## Introduction

**Ruby In Steel** is a Ruby IDE for Microsoft Visual Studio 2008 or 2005. Developed by SapphireSteel Software, it is available in two principal editions:

### Ruby In Steel Developer

Aimed at professional programmers, **Ruby In Steel Developer** is a powerful editing and debugging environment for Ruby and Ruby On Rails. Its intelligent inference engine provides numerous Ruby IntelliSense features; it includes the ultra-fast '*Cylon*' debugger and SapphireSteel Software's unique drag and drop Ruby On Rails design environment, *The Visual Rails Workbench*.

### Ruby In Steel Text Edition

**Ruby In Steel Text Edition** provides project management, editing and debugging tools for Ruby and Rails development and includes integrated debugging (but not the fast *Cylon* debugger). It may either be installed 'standalone' (in which case it also installs a 'Ruby-flavor' version of Visual Studio 2008) or it may be integrated into an existing version of Visual Studio 2008 if one is present on your PC.

### Other Editions

From time to time, Ruby In Steel may also be made available in 'Personal' editions. These may offer a subset of the features of one of the editions above. For example, *IronRuby In Steel 'Personal Edition'* contains only those features needed to develop programs using Microsoft's IronRuby for .NET. Not all the features described in this manual apply to these editions.

### This Manual

This is the official reference manual and user guide to Ruby In Steel. **Note that not all the features are available in the Text Edition.**

## The Installation Guide

Before installing Ruby In Steel, be sure to read the Installation Guide, **Installation Instructions.pdf**, supplied with the software.

To buy or download the latest versions of both the Ruby In Steel software and this manual and to keep up to date with news and tutorials, be sure to visit the SapphireSteel web site:

<http://www.sapphiresteel.com>

## Installation

**Requirements:** You must have Visual Studio 2008 or 2005 (Standard edition or higher), Windows XP (service pack 2) or Vista and a Ruby interpreter (**ruby.exe**). Alternatively, you may optionally install a single-language (Ruby) version of Visual Studio 2008 at no additional cost using our 'All-in-one' installer.

### How To Install Ruby In Steel

**IMPORTANT!** If you need to install a free Ruby language edition of Visual Studio 2008, a Ruby interpreter, the Rails framework, the MySQL database or any combination of these components, you can do so using our 'All-in-one' installer available on the [Ruby In Steel Download page](#).

### The Installation Guide

If you need help choosing which components to install, please read the separate **Installation Guide** (PDF format) which is supplied with the software. This explains every step of the installation procedure in detail.

### Quick Install...

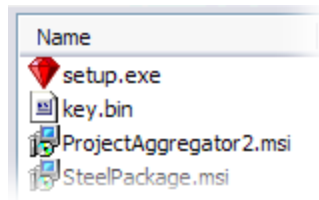
If you have a previous version of Ruby In Steel installed, be sure to remove this first: Select *Start Menu*, then *Settings, Control Panel, Add Or Remove Programs*. Finally, select the Ruby In Steel item and click the *Remove* button.

Unzip the Ruby In Steel installation archive into a directory of your choice (for example, C:\temp). This directory will now contain the files needed for the installation of Ruby In Steel.

Make sure that Visual Studio is not running when installing (or uninstalling) Ruby In Steel.

## FIRST, INSTALL YOUR KEY

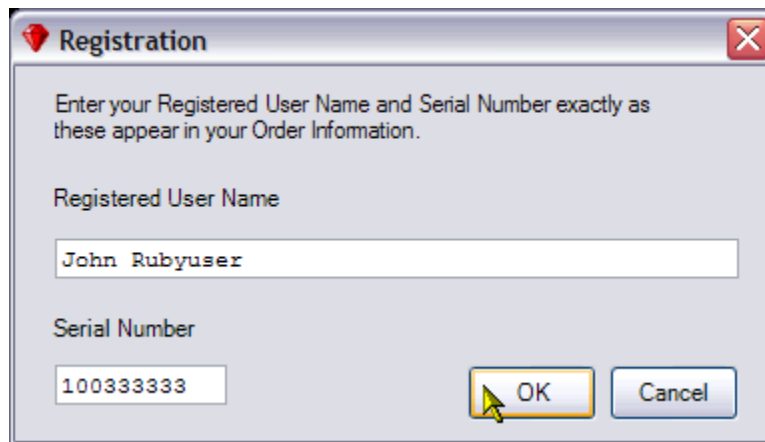
If you have a licensed edition of Ruby In Steel, you should have received or downloaded a 'key file' with your order. The key file is named **key.bin**. This must be copied into the same directory as **Setup.exe**. For example, if you unzipped the installation files (including **Setup.exe**) into *C:\temp* you must copy **key.bin** into *C:\temp* before installing Ruby In Steel.



## If you are installing Ruby In Steel into an existing version of Visual Studio...

First ensure that you have downloaded the appropriate version of the installer (either for Visual Studio 2005 or 2008) then follow these steps...

Double-click **Setup.exe** to begin installation. After a few moments you will see the Ruby In Steel Installer Splash screen. Select all the options you wish to install and follow all directions throughout the setup process. The Ruby In Steel installer displays this registration dialog:



In the Registration Dialog, you should enter the following details:

- **Registered User Name**  
This is the name that the product is licensed to (without quotation marks).
- **Serial Number**  
This is the Order number which will have been sent with your order conformation.

You must enter these details EXACTLY (uppercase and lowercase characters, punctuation and spaces are significant). Click OK to confirm registration details.

If an error message appears, please verify that you have entered the user name and serial number correctly and that the file, **key.bin**, has been copied into the directory containing **Setup.exe**.

When complete you will see a screen stating '**Installation Complete**'. Press the *Close* button and you are now ready to load Visual Studio with Ruby In Steel.

## Checklist On First Using Ruby In Steel

When you load Ruby In Steel for the first time, you should check that the paths to the Ruby Interpreter and, optionally, to your database server(s), are correct:

See '[Configure Ruby and Database Paths](#)'.

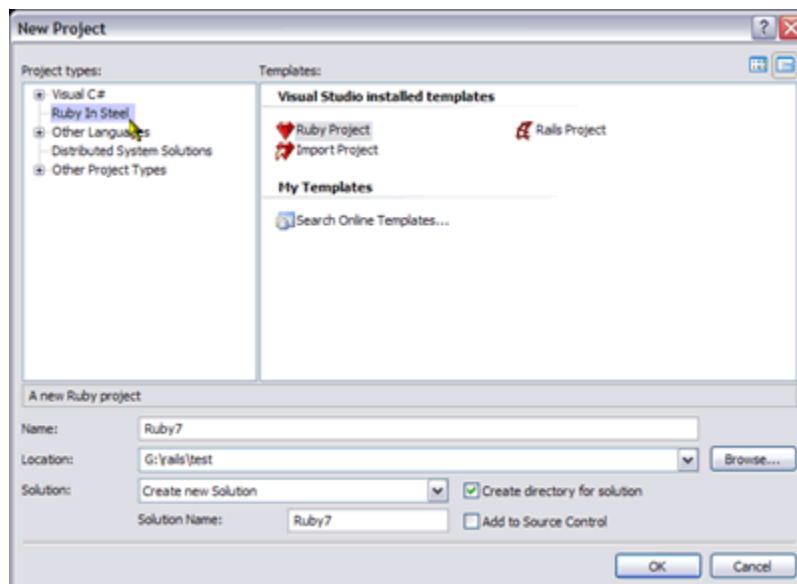
## Five Minute Guide To Ruby In Steel

IF THIS IS THE FIRST TIME YOU'VE USED RUBY IN STEEL, THIS STEP-BY-STEP WALKTHROUGH WILL EXPLAIN HOW TO CREATE AND RUN YOUR FIRST PROGRAM IN A MATTER OF MINUTES...

### Create a New Project

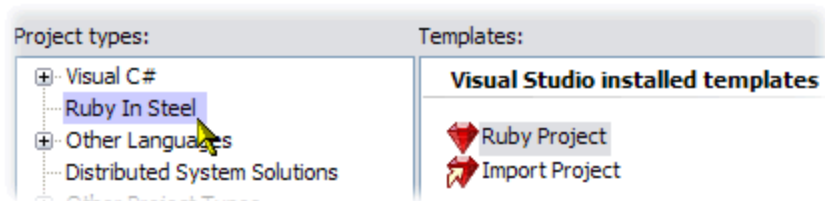
You can start a new Ruby In Steel project just as you would start any other Visual Studio project.

Select the *File* menu, then *New*, then *Project*.



The New Project Dialog

When the New Project dialog appears, select the *Ruby In Steel* branch in the left-hand pane. For now we shall create a plain vanilla Ruby project so make sure the *Ruby Project* icon is selected in the right hand window.

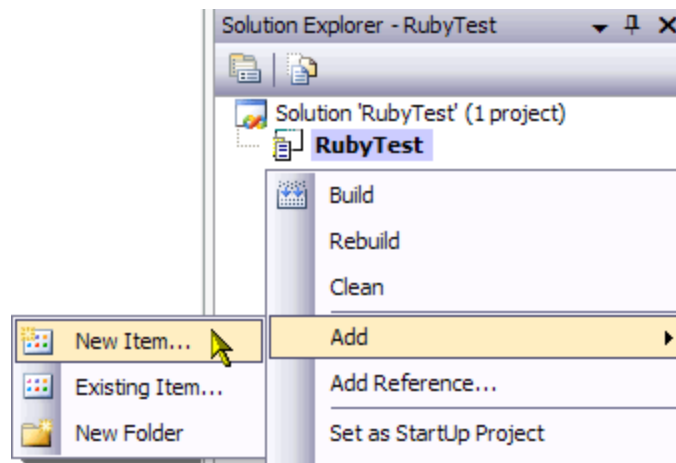


Give the project a name by filling out the *Name:* field at the bottom of the dialog (for now enter the name *RubyTest*). If you wish, you may click the *Browse* button to select a directory, otherwise you may accept the default location. If you check the '*Create directory for solution*' button, a new subdirectory will automatically be created for your project. Now click *OK*.

### Projects and Solutions...

Ruby In Steel organizes Projects in the form of branches in a Solution. A project is a group of one or more files in one or more directories. A Solution is a group of one or more projects. You can add new projects by right-clicking the Solution at the top of the Solution Explorer and selecting *Add, New Project*. The Solution Explorer provides a convenient way of keeping related files grouped together from one work session to the next. For Rails programmers, it also gives you an easy way of working with the numerous folders comprising a Rails application.

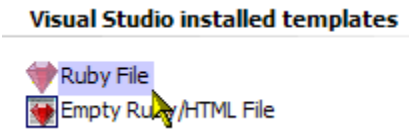
When you create a new Ruby project, an empty source code file, **rubyfile.rb**, is added automatically. Optionally, you may add more Ruby code files to the Project using the Solution Explorer. Let's do that now: right-click the indented Project branch (not the top-level Solution itself) and select *Add, then New Item...*



When adding a new Ruby file, right-click the indented Project branch and select *Add, New Item*

In the *Add New Item* dialog, select the *Ruby File* item (this will create an empty Ruby file) and give it the name, **test.rb**, in the field at the bottom of the dialog. Then click *Add*.





Now you are ready to start writing your program. Enter the following:

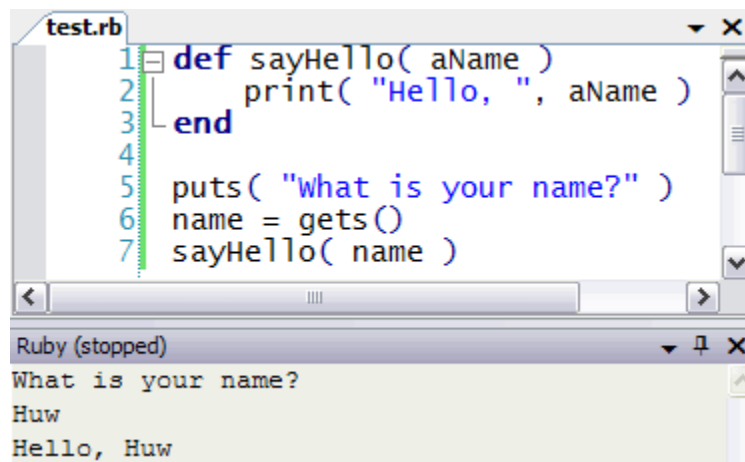
```
def sayHello( aName )  
    print( "Hello, ", aName )  
end  
  
puts( "What is your name?" )  
name = gets()  
sayHello( name )
```

Notice that Ruby In Steel automatically color codes the text as you enter it so that reserved words such as **def** and **end** are obviously different from method names such as **print** or **sayHello** and strings such as "Hello". It also 'outlines' your code so that the body of a method such as **def sayHello** can be hidden from sight by clicking the minus symbol in the 'button' to the left of its name or made visible again by clicking the plus symbol.

### Code Collapsing

Ruby In Steel has advanced 'code collapsing' capabilities. It collapses methods, modules, classes, **if..else** blocks, **case** statements, **for**, **while** and **until** loops, blocks delimited by { and } or by **do** and **end** - and more besides...

To run the program , press CTRL+F5. If there is a syntax error a window pops up to tell you. Click the error message to locate the problem code in the editor. Fix the error and try again. When any syntax errors are fixed, press CTRL+F5 again and your program will run inside an interactive integrated console window, seen below the editor here...



## Project Management

RUBY IN STEEL LETS YOU CREATE, IMPORT AND MANAGE COMPLEX PROJECT GROUPS COMPRISING NUMEROUS FILES AND FOLDERS.

### Create A New Project

Ruby In Steel organizes your programs in the form of one or more projects grouped together as a Solution. To make use of all the editing and debugging features, you must create a project to which your Ruby files are added.

### How To Create A New Ruby Project

- > 1. Select the *File* menu, then *New, Project* (or simply press *CTRL+SHIFT+N*).
- > 2. In the New Project Dialog select the Ruby In Steel Project type in the left pane.
- > 3. Select the *Ruby Project* icon in the right pane.
- > 4. Enter a name for the new project and optionally browse to find an empty directory.
- > 5. You may optionally select *Create directory for solution*.
- > 6. Click *OK*.

### Importing and Converting Projects

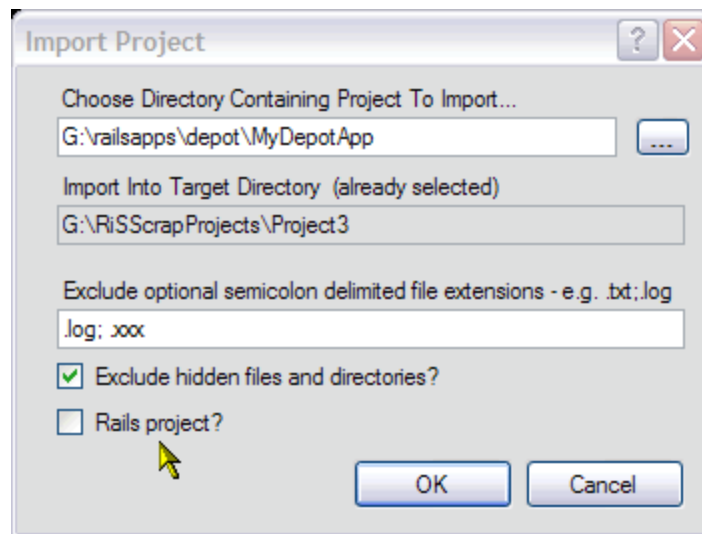
If you have an existing Ruby or Rails project that you wish to import into Ruby In Steel, you may either make a complete new copy of the existing project in a new directory on disk or you may convert the project 'in place' (Developer Edition only) by creating a Visual Studio Solution in the top-level directory of the original project.

A 'project' may be any group of files and directories on disk. In the case of a Rails application, the top-level directory of a project is normally the folder whose name identifies the application (e.g. **\MyBlog**) and which contains the various Rails directories such as **\app** and **\config**. When imported into Visual Studio, the files and directories will be added into a Visual Studio 'Solution file' and they will then appear beneath a Project branch in the Solution Explorer.

## How To Import A Project

### FOLLOW THESE STEPS TO COPY FILES TO A NEW LOCATION AND IMPORT THEM INTO A VISUAL STUDIO SOLUTION...

- > Select the *File* menu, then *New, Project* (or simply press *CTRL+SHIFT+N*).
- > In the New Project Dialog select the *Ruby In Steel Project* type in the left pane.
- > Select the *Import Project* icon in the right pane.
- > Enter a name for the new project and optionally browse to find an empty directory.
- > You may optionally select *Create directory for solution* (this is recommended).
- > Click *OK*.
- > The Import Project dialog now appears...



- > Here the first field is the *source directory* of the Project you wish to import. You can click the button to the right of this field to browse and find the directory containing this project.
- > The second field is the *target directory* (which you previously chose in the New Project dialog) into which the old project will be copied in order to create your new Ruby In Steel project. This field is read-only. The target directory must not be the same as the source directory.
- > In the third field you may optionally specify one or more extensions of file types which should not be imported. The file extensions should be entered following a period and separated by semicolons like this:  
**.log;.tmp;.xyz**
- > When the 'Exclude hidden files and directories' *Check Box* is selected, Hidden files and hidden directories will not be added to the imported project.

- If this is a Ruby On Rails project, be sure to check the '*Rails project?*' checkbox. This will enable certain Rails-specific features for the imported project such as the One-Click Rails Debugger.
- Before the project is imported, a '*Creation and Import Data*' dialog is displayed. This shows the following information:
  - Total number of files to be copied*
  - Solution name*
  - Solution directory*
  - Import from directory*
  - Database*
- You should verify that these details are correct. In particular, check the number of files which will be copied. If you have selected more files than you actually intended (by accidentally selecting the wrong source directory, perhaps), now is your last chance to back out.
- If all the details are correct, click the *Proceed* button to begin the import process.

Ruby In Steel will copy the entire directory structure and all the files it contains to the *Target* location of the new project. Any files which you have specifically excluded *will* be copied to the target directory but *will not* be added to the Solution Explorer.

## Notes on Importing Projects

- 1) We recommend that you always *create a new directory for an imported project*. We caution against attempting to copy files into a directory which already contains source files. The project importer will overwrite any similarly named files and directories in the target directory.
- 2) The 'top level' directory of your existing *Rails* project (that is, the one you import *from*) will normally be the directory which 'names' the project (e.g. **\Blog**) and which contains various subdirectories such as **\app**, **\components**, **\config** and so on. When making a copy of a project, the Importer copies all the files in a Rails project into the new target directory. It will not make any changes to the database or configuration files.
- 3) If you have created solution files using previous versions of Ruby In Steel, you may find that the old solution files do not display correctly in the Solution Explorer. Use the Project Importer to convert them for use with the latest version of the software.
- 4) The Solution Explorer cannot include file (and path) names with certain special-purpose names or XML-specific characters such as '&' or single-quotes. The Project importer will copy such files to the selected Target directory but will not add them to the Solution Explorer. Omitted files will be listed in the *General* page of the Visual Studio *Output* window. You may view all files (including those excluded from the project, by selecting *Project, Show All Files*).

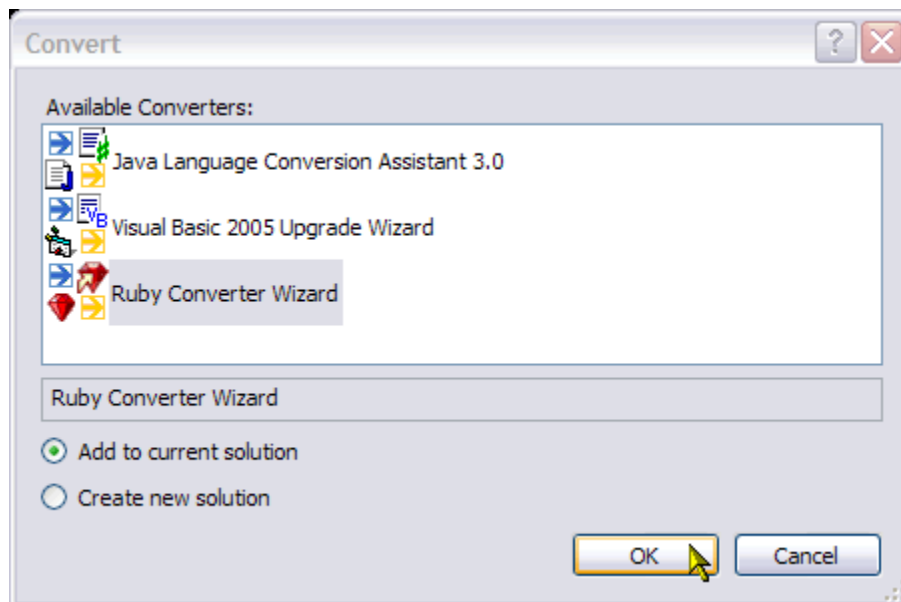
## How To Convert A Project

### [Developer Edition Only]

FOLLOW THESE STEPS TO CREATE A VISUAL STUDIO SOLUTION IN THE SAME DIRECTORY AS AN EXISTING PROJECT...

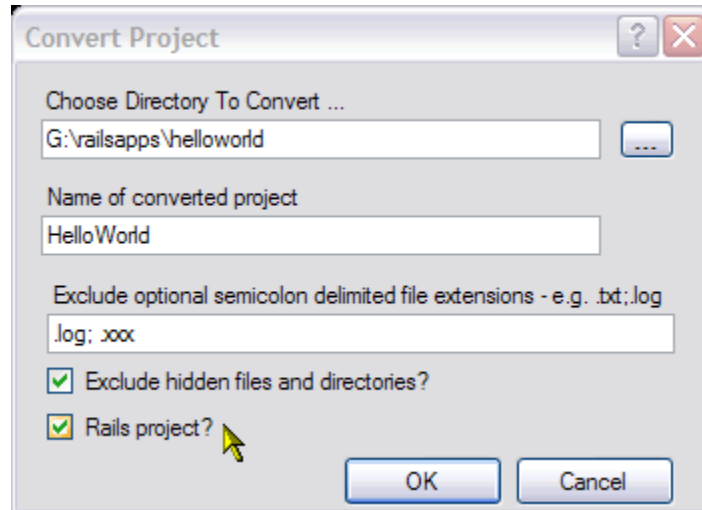
When a project is 'converted', existing files and folders are added to the Solution Explorer. No modifications are made to the files themselves. The Convert option may not be available in some configurations of Visual Studio. In that case, use *Import*.

- > Select the *File* menu, then *Open* and *Convert...*
- > In the *Convert* dialog, select *Ruby Converter Wizard*



- > Click *Create New Solution* if you wish to create a Solution specifically for this project. If you already have a Solution open in Visual Studio and you wish to add your Ruby solution as a Project in that solution, click *Add to Current solution*. Click OK

The *Convert Project* dialog appears. Here you must choose the directory containing the Ruby (or Rails) files and directories which you wish to convert. You may click the Browse button [...] to select a directory.



- > In the second field you should give a name for the converted project.
- > In the third field, you may optionally specify one or more extensions of file types which should not be imported. The file extensions should be entered following a period and separated by semicolons like this:  
**.log;.tmp;.xyz**
- > When the '*Exclude hidden files and directories*' checkbox is selected, Hidden files and hidden directories will not be added to the imported project.
- > If this is a Ruby On Rails project, make sure the '*Rails project?*' checkbox is ticked. This ensures you will be able to use Rails-specific features such as the One-Click Rails debugger. Leave it unticked for other types of Ruby project.
- > Click OK to convert the project and display the files in the Solution Explorer.

## Using The Solution Explorer

THE SOLUTION EXPLORER IS YOUR CONTROL CENTER FOR RUBY AND RAILS PROJECTS. HERE YOU CAN GROUP TOGETHER NUMEROUS RELATED FILES AND FOLDERS AND ADD, REMOVE OR RENAME THEM AS REQUIRED.

### To Add A New Project

Right-click the (top-level) *Solution* item. Select *Add, New Project*. Select the Project type from the Add New Project Dialog and continue as explained earlier on how to create or import a project.

### To Add A New File

Right-click a *Project* branch (indented beneath the top-level *Solution* item). Select *Add, New Item*. Now select a file type (*Ruby File* creates an empty Ruby **.rb** file; *Empty ERb File* creates an empty Rails 2 **.html.erb** template file; *Empty RHTML File* creates an empty Rails 1 **.rhtml** template file). Various other types of file such as HTML, XML and text files can also be added.

### To Add An Existing File From Disk

Right-click a Project branch. Select *Add, Existing Item*. Select a file on disk. Click *Add*.

### To Add A New Folder

Right-click a Project Branch. Select *Add, New Folder*. This creates a subdirectory in the Project directory.

### To Rename A File Or Folder

Select the File or Folder in the Solution Explorer. Press F2. Enter a new name.



## TO EXCLUDE A FILE OR FOLDER FROM THE PROJECT

Right-click a File or Folder. Select *Exclude From Project*. This removes the item or items from the project but does not delete them from disk.

## TO DELETE A FILE OR FOLDER

Right-click a File or Folder. Select *Delete*. Or press the Delete key on your keyboard.

**Caution:** Deleting a file or folder from the Solution Explorer will also delete the file or folder on disk!

## DRAG-AND-DROP TO COPY OR MOVE

You may select files or folders and move them to other locations in a solution by dragging with the mouse. To copy rather than move files, hold the CTRL key while dragging.

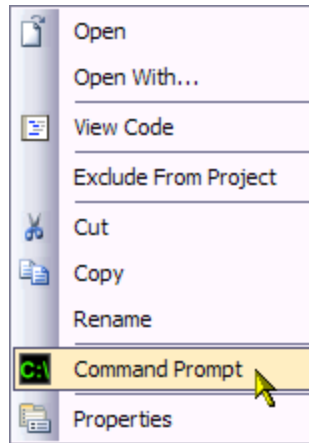
## TO SHOW ALL FILES

Normally the Solution Explorer does not display files which are not included in a Ruby or Rails project but may, nonetheless, be present on disk (these include the Visual Studio and Ruby In Steel solution and project files – the extensions *.sln*, *.suo* and *.stproj*), files with the extension *.log* and files which you have specifically excluded from the project. You can force the display of all files in the application directories by selecting *Show All Files* from the *Project* menu; this also expands all branches of the Solution Explorer. Click the menu item again to resume the default view. The Solution Explorer itself also has a small *Show All Files* button (when a project node is selected) beneath its caption bar.

## TO SYNCHRONIZE

If files have been added or deleted from a project's directory outside of Visual Studio, the display of the files in the project can be brought up to date by clicking the Synchronize button beneath the Solution Explorer's caption bar. For more information, see: [Synchronize](#).

## TO OPEN A COMMAND PROMPT IN A DIRECTORY



With Ruby In Steel there is no need to navigate to a directory from the system prompt. Just right-click a file or folder in the Solution Explorer and select *Command Prompt* from the menu. This opens a command window in the selected directory.

## File Properties

You can select a file in the Solution Explorer and set its properties in the Properties panel. These properties can be used to configure certain operations affecting the way in which the file will be run or built. The available properties are shown below:

Property	Description
Arguments	Arguments to pass to your program
BuildAction	How to build/run this file
Flags	Flags to pass to <i>Ruby.exe</i> such as <b>-help</b> , <b>-w</b> or <b>--version</b>
Load Paths	Specify <i>\$LOAD_PATH</i> ; this is a semicolon delimited list (equivalent to the <b>-I</b> flag)
Require Files	a semicolon delimited list of required files (equivalent to the <b>-r</b> flag)
Working Directory	The directory in which the Ruby file will run

### NOTES ON BUILD ACTION:

The default BuildAction for Ruby (*.rb*) files is **Ruby**; and for a Rails (*.html.erb* or *.rhtml*) template files it is **EmbeddedRuby**. For other file types it may be **None** or **Content**. Other options may be displayed but currently have no effect and are reserved for future use. You may change these build actions as required. These are the build actions available:

**Ruby** – checks the Ruby syntax of the file.

**EmbeddedRuby** – processes the file using ERB, the embedded Ruby processor.

**Content/None** – takes no action.

## Project Properties

A number of settings can be applied to the current project using the *Project Properties* Pages (available from the *Project* menu).

**Note:** If you do not specify Project properties, the properties of the current project will automatically adopt the global options (available via *Tools/Options*). When specified, however, Project properties take precedence over global properties. In addition, file properties (set in the Properties page for a file selected in the Solution Explorer) take precedence over Project properties. **Not all properties are available in all editions of Ruby in Steel.**

There are two pages of Project Properties:

### GENERAL

This page specifies the defaults for the current project. These properties are used with all configurations for this project.

### BUILD

This page allows you to modify and save named configurations which may be selectively loaded in the current project. For example, you might save three named configurations: *Debug*, *Test* and *Release*, each of which has different properties for items such as the Ruby Interpreter, the Ruby Script Arguments or the Rails Server Port. When you wish to switch from one configuration to another, you may select the configuration name from the drop-down 'Solution Configurations' list just beneath the Visual Studio main menu.

## **To Create a New Project Configuration**

- Select Build, Configuration Manager.
- Click the Active solution configuration drop-down.
- Click <New...>
- Enter a name for this configuration.
- Click OK.
- Click Close.

## **To set Build Properties for a Named Configuration**

- Select Project, Project Properties.
- Select the Build tab.
- From the Configuration drop-down select a Configuration name.
- Change any properties for this configuration and click File, Save All.

## General Page Project Properties

### DEBUGGER

- > Allow Editing While Debugging

When true allows the code to be edited when debugging. Otherwise code file is set to read-only

### AUTOS WINDOW OPTIONS...

The following options determine which details will be displayed in the Autos window when debugging. Be aware that there is a trade-off between completeness and speed. When all options are enabled, there may be a noticeable delay in evaluating large data structures. This is particularly true in Rails when - for example - the evaluation of **self** may involve huge amounts of data. In general, it is better to use the Watch windows to monitor specific variables.

- Autos Window (Class Variables)
- Autos Window (Global Variables)
- Autos Window (Instance Variables)
- Autos Window (Object Methods)
- Autos Window (self)
- Autos Window (Singleton Methods)

- > Enable Just-In-Time Debugging

When true, an exception will trigger the debugger.

- > F5 Starts Ruby

When true in a Rails project, F5/CTRL-F5 debug/run current Ruby file. When false, they debug/run the Rails application.

- > Keep Stack Frames

When true, stack frames are stored to allow navigation up and down the stack using the Call Stack window. When false, debugging is slightly faster but there is no Call-stack navigation.

- > One Breakpoint Per Line

When true, a line containing multiple expressions will cause a single step when debugging.

- > Substitute '/' for '\'

If some breakpoints are not hit, this may be due to inability to match path names due to the mix of '/' and '\' path separators. Turn this on to fix this problem. Note that this slows the debugger so should not normally be enabled by default.

> Use the Cylon Debugger

When true, the fast Cylon debugger will be used (in those Ruby In Steel editions where it is available). When false, the much slower Ruby debugger is used.

## RAILS

> ERb Flags

Any flags to pass to the ERb (Embedded Ruby) processor.

> ERb Library Files

Library files to be used by ERb (a semicolon delimited list).

> ERb Processor

The path to the ERb processor (if you wish to override the default).

> ERb Timeout

The time in seconds allowed for the ERb processor to complete.

> Framework

The Rails framework version (e.g. *Rails1* or *Rails2*). This property determines the default extension used for Rails template files.

> Rails Project

*True* for a Ruby On Rails Project. Used for **require** files, IntelliSense and debugging.

## RUBY

> Default Ruby Interpreter

The path to the default Ruby interpreter (e.g. **ruby.exe**).

## SYNCHRONIZATION

> Exclude Directories

A line-delimited list of directories to omit when synchronizing.

> Exclude Extensions

A semicolon-delimited list of file extensions to omit when synchronizing. e.g.

**.txt; .svn; .log**

> Exclude Files With No Extension

When True files that have no filename extension are not synchronized.

> Exclude Hidden Files and Folders

When True, files and directories with the Hidden attribute set are not synchronized.

## VISUAL RAILS WORKBENCH

- > Automatically Update Files

This determines whether changes in the active page design (e.g. changes that affect shared layouts and partials) are synchronized in any other open page designs when changes made to the active page design are 'committed' - otherwise they will not be synchronized.

- > Design time image path

The path to which to append non-rooted image files in <IMG> tags at design time. This assists in displaying images in a specific path in the page designer.

- > Diff/Merge Tool

An (optional) difference/merge tool to use when comparing backup files and archive file with the current ERb or page layout document.

- > Diff/Merge Tool Parameters

Any parameters which you wish to pass to your Difference/Merge tool.

- > Page Design Style

An optional style block which can be used to define design-time styles to help highlight the partial (**.erb-partial**) and view (**.erb-view**) components of a composite page design in the Visual Rails Workbench. For example, the following styles will outline a view in a 2 pixel dashed border in Navy and partials in a 2 pixel dotted border in Green:

```
<style title="erb-style" type="text/css" xmlns=""><!--  
/* SapphireSteel RWB styles */  
.erb-view{ border: 2px dashed; border-color:Navy; }  
.erb-partial{ border: 2px dotted; border-color:Green; }  
--></style>
```

- > Use Visual Rails Workbench

When *true*, the Visual Rails Workbench integrated environment will be activated when you edit Rails template files. When *false*, a simple ERb editor will be used.



## Build Page Project Properties

### BUILD EVENTS

- > Post-Build Event  
a command to run after the 'build' completes.
- > Pre-Build Event  
a command to run before the 'build' commences.

### MISC

- > Working Directory  
The directory in which IRB, Rails, Generate and Rake commands are executed.

### RAILS

- > Rails Script  
The Ruby script used to start Rails from within Visual Studio.
- > Rails Server  
The Server to be used when running a Rails application (you must ensure that the selected server has been installed). Selecting an item here will automatically set the Rails Script and Web Server Script properties.
- > Rails Server Port  
The port (e.g. 3000) to be used for the Rails server (applied to Mongrel and WEBrick only).
- > Web Server Script  
The script used to start the web server.

**The Rails Server and Scripts.** The *Rails Server* property determines which server Rails Server and web Server startup files to run. For example, selecting *WEBrick* will cause the file **webbrick\_service.rb** to be selected as the Rails Script and the file **webbrick\_server.bat** as the Web Server Script. If necessary you may edit these files in order to change the Ruby script and the path to the server. Note that some startup files refer to files located in your Ruby installation and may require specialist knowledge in order to change the default values. The server startup files are located in the **\scripts** directory beneath your Ruby In Steel Installation.

## **RUBY**

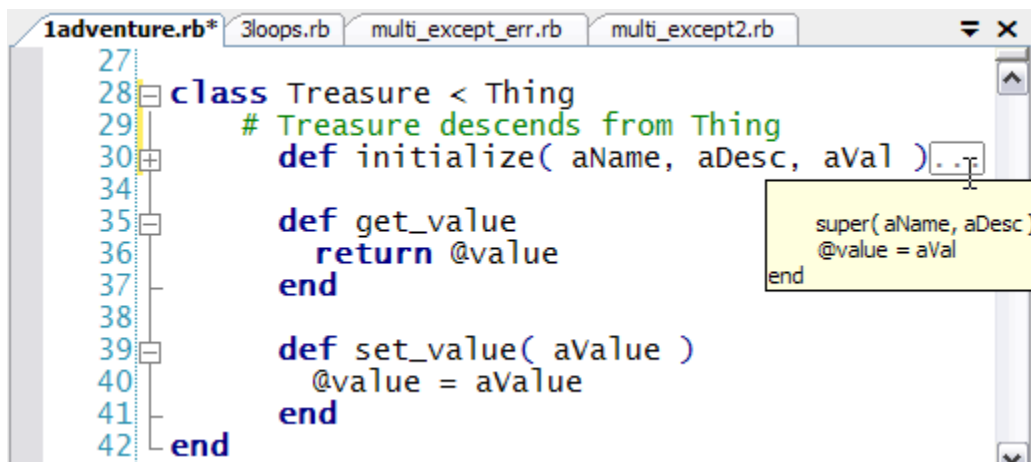
- > Ruby Interpreter  
The path to the Ruby interpreter (e.g. **ruby.exe**).
- > Ruby Interpreter Flags  
Any flags (e.g. **-w** or **-version**) to be passed to the Ruby interpreter.
- > Ruby Library Files  
Library files to be used by Ruby (a semicolon delimited list).
- > Ruby Library Paths  
Paths to library files (a semicolon delimited list).
- > Ruby Script Arguments  
Any arguments to be passed to a Ruby program.
- > Ruby Start File  
The file that Ruby runs (if left blank, Ruby runs the currently active Ruby file).
- > Ruby Type  
Determines which type of Ruby interpreter/compiler is being used when debugging.  
(e.g. Standard Ruby or JRuby).

## The Editor

THE RUBY IN STEEL EDITOR BUILDS UPON THE EDITING CAPABILITIES OF VISUAL STUDIO AND ADDS 'RUBY SPECIFIC' FEATURES...

### Overview

The Ruby In Steel editor provides enhanced code-coloring and collapsing for Ruby and Rails files plus automated commenting/uncommenting, multi-level undo/redo, drag-and-drop copy and paste, code indenting, incremental search with regular expressions, code formatting and much more...



Ruby In Steel provides syntax sensitive code coloring and collapsing

### Syntax Coloring

#### > Ruby (.rb files)

Ruby In Steel automatically colors Ruby language elements including keywords, comments and data types such as strings, floats and integers and Ruby-specific elements.

#### > Rails (.erb and .rhtml files)

Ruby In Steel colors html elements such as tags, strings and comments. In addition, the language elements of embedded Ruby code are also colored.

#### > Configure Coloring

The colors can be altered to taste. Select the *Tools* menu then *Options*. In the *Options* dialog, select *Fonts and Colors*. The Visual Studio colors for elements common to all languages (e.g. Keyword or String) are used by Ruby In Steel. There are also some language specific coloring options which are preceded by the names **Ruby** (e.g. *Ruby Brace*) or **Rails** (e.g. *Rails Attribute*).

## Code Collapsing

### > Automatic 'outlining' / code 'folding'

Ruby In Steel outlines your code so that you can selectively hide code blocks such as classes, modules, methods and programming constructs such as **if** and **while** blocks. In **.rhtml** files, code collapses on matching tags such as **<body></body>**. Note that collapsing is dependent upon correct syntax. In **.rhtml** files, matching pairs of opening and closing tags are required. Click the plus (+) or minus (-) symbols to toggle code collapsing.

Ruby elements which are auto-collapsed include:

- classes
- modules
- methods
- **case** statements
- **if** blocks
- **while**, **unless**, **until**, **for** loops
- **=begin/=end** comment blocks
- blocks { and } or **do** and **end** delimited

### > Optional 'mark and collapse areas'

If code blocks aren't automatically outlined, you can mark and collapse any code you wish using the mouse. This lets you hide 'free standing' code areas or comment blocks, for example.

#### > View Tooltip of Collapsed Area

For a fast view of the hidden text in a collapsed area, hover your mouse over the three dots indicating a collapsed area [...]. This will display the hidden code in a tooltip.

#### > Collapse To Definitions

To collapse to all 'top-level' outlines (e.g. all the Class headers in a file containing many classes) right-click in the editor and select, *Outlining, Collapse To Definitions*.

#### > Hide Selection

You can optionally collapse areas which are not normally collapsed (e.g. a comment block). Mark the area using the mouse then right-click and select *Hide Selection*.

Use the other options on the *Outlining* menu (right-click in the editor) to toggle the outlining display throughout the current file.

## Bracket Matching

Ruby In Steel has automatic bracket matching and highlighting. Scroll your cursor over a bracket in order to highlight a pair of matching brackets. Press **CTRL+]**  to move the cursor quickly from one matching bracket to another.

```
{
  'name' => 'Multi-Hash',
  'array' => ['one', 'two', 'three', 'four'],
  'nested array' =>
    [
      "I",
      ["wandered", "lonely", "as",
        ["a", "cloud"]
      ]
    ],
  'nested hash' => {'a'=>'hi', 'b'=>'goodbye'}
}
```

## Keyword..end Matching

Many Ruby constructs are delimited by an opening keyword and a closing **end** – for example: **class..end**, **module..end**, **def..end**, **while..end** and **until..end**. You can move your cursor between an opening keyword and the matching **end** using the same key combination used for bracket matching: **CTRL+]** .

```
6 module MyModule
7   class MyClass
8     def x( s )
9       puts( s )
10    end
11  end
12 end
```

*Here we press **CTRL+]**  to toggle between a class definition...*

```
6 module MyModule
7   class MyClass
8     def x( s )
9       puts( s )
10    end
11  end
12 end
```

*...and its matching **end** keyword.*

## Commenting/ Uncommenting

Mark a block of code and click the *Comment* button (or press **CTRL+E, C**) to comment out the block. To uncomment a block, click the *Uncomment* button on the Text Editor Toolbar (or press **CTRL+E, U**). Commented blocks in Ruby source files are marked by inserting comment characters **#** at the start of each line. In Rails template files, HTML comment delimiters are used: **<!--** and **-->**.

## Smart and Block Indenting

Three code indenting styles are supported. These may be set in the *TextEditor/Ruby/Tabs* page of the *Options* dialog. They are:

- **None** – no automatic indenting.
- **Block** – indent code align with preceding line.
- **Smart** – attempt to align by syntax (e.g. align **end** with matching **def**).

## Automatic Code Formatting

You can reformat an entire Ruby document or a selected block of code to regularize the indenting automatically.

### FORMAT DOCUMENT

The entire contents of a Ruby code file can be auto-formatted by selecting *Edit, Advanced, Format Document* (default shortcut: **CTRL+E,D**)

### FORMAT SELECTION

The currently selected block of Ruby code can be auto-formatted by selecting *Edit, Advanced, Format Selection* (default shortcut: **CTRL+E,F**)

## Other Features

Ruby In Steel provides the same rich set of editing features available to other Visual Studio languages including: Multiple level undo/redo (**CTRL+Z**; **CTRL+Y**), Indent/Outdent, bookmarks and macros.

## Intelligent Coding Tools

### [ Most Features Developer edition only]

Ruby In Steel Developer provides a number of coding aids to assist in the writing of code. These include a variety of IntelliSense tools such as 'completion lists', 'quick info' and 'parameter info' as well as in-code syntax error marking ('wavy lines') and auto-expanding Ruby code 'snippets'.

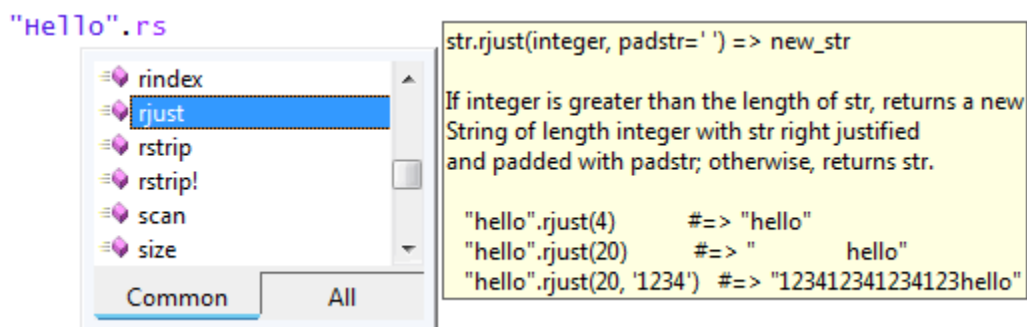
### IntelliSense

Ruby in Steel implements the following IntelliSense features for Ruby source files:

- Member Completion Lists
- Keyword Completion Lists
- Variable Completion Lists
- Quick Info Tooltips
- Parameter Info Tooltips
- Snippets

### Member Completion Lists

Member completion lists are drop-down lists showing the available methods, modules and classes which may be used after a completion character is entered into the code editor. A completion character may be either a single dot ( . ) or a double colon ( :: ) as dictated by Ruby syntax. Where embedded documentation is provided (in the form of RDoc comments in the source code), this documentation is displayed as a tooltip when an item is selected in the completion list.



Ruby In Steel tries to infer the type and determine the scope of an identifier in order to display the appropriate completion list. However, bear in mind that Ruby is a dynamic language and the formal types of identifiers are not predeclared so it may often be impossible for Ruby In Steel to 'guess' the type of some variables.

When a specific type cannot be inferred, the completion list defaults to members of the Object class. You may, however, specifically assert that a certain type should be assumed in order to generate better IntelliSense. Refer to the section on Intelligent Type Inference, at the end of this chapter, for more information.

## Common and All Tabs

Often the Completion list displays two tabs - Common and All. The Common Tab generally displays members of the current object's class and any immediate ancestor classes. The All Tab displays all the members shown in the Common Tab plus the members of the Object class. Once you have selected a tab, that tab will remain the default until the other tab is selected.

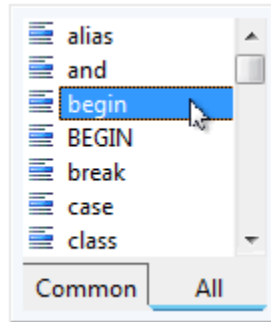
When an object is an instance of Object (or when the IntelliSense is unable to infer any specific descendent class for a variable), an untabbed completion list is displayed showing members of Object.

### Configuring Completion Lists

Optionally you may turn off the display of Object methods and Ancestor in order to restrict the completion lists to the members of more specific classes. The *Display Object Methods* and *Display Ancestor Methods* option can be set in the *Text Editor, Ruby, IntelliSense* page of the *Tools, Options* dialog.

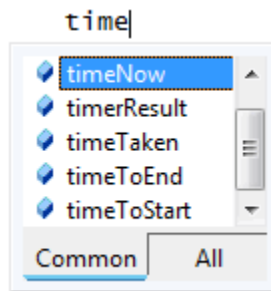


## Keyword Completion Lists



Ruby keywords such as **def**, **while**, **module** and **class** may be selected from a completion list. This list appears when matching characters are entered if *Show completion list after a character is typed* is selected in the IntelliSense Options (for example, if you enter '**mo**', the completion list will highlight '**module**'). You can also force the completion list to appear by pressing CTRL+Space. Keywords are displayed in the *All* tab. If a keyword match is unambiguous, it may also be displayed in the *Common* tab.

## Variable Completion Lists



The completion list will suggest variable names (including local, instance or class variables) matching text that has been typed. Force the display of the completion list by pressing CTRL+Space.

## Quick Info Tooltips

```
myhash| = Hash.new
myhash.values
```

Method values  
hsh.values => array

Returns a new array populated with the values from hsh. See also Hash#keys.

```
h = { "a" => 100, "b" => 200, "c" => 300 }
h.values #=> [100, 200, 300]
```

When the source code of a class or method is preceded by a comment block, the comments will be displayed in a tooltip when the mouse hovers over a class or method name in the code editor. When an object of a known class has been created, its tooltip displays the name of its class. Optionally this feature may be disabled by deselecting *Display RDOC In Tooltips* in the *Options* dialog. Documentation may also be shown in the [RDOC Window](#).

## Parameter Info Tooltips

```
calcTax(|
```

```
calcTax (subTotal:Object, taxRate:Object): Float
```

Information on the parameters expected by a method can be displayed in a tooltip when you are calling that method. The tooltip is triggered by the opening bracket ( and the highlight moves from one parameter to the next as the separating commas are entered. The return type of the method is also shown. By default all parameters are shown as instances of the Object class. If the return type can be inferred, its class is shown, otherwise NilClass is shown. To enable more information in Parameter Info tooltips, see *Type Assertions*, later in this chapter.

## Snippets

Snippets are ready-to-use code fragments which can be inserted at the cursor position in the code editor by entering a shortcut or selecting a named snippet from a menu. Ruby In Steel provides a library of Ruby snippets some of which include interactive fields or 'replacement points' into which you can enter specific values such as the names of variables.

### TO INSERT A SNIPPET FROM A MENU

Right-click at the point in the editor where you would like to insert the snippet. Click *Insert Snippet* on the shortcut menu. Scroll down to the name of the desired snippet and double-click it or press TAB to insert the snippet into your code.

### TO INSERT A SNIPPET FROM A SHORTCUT

Snippet shortcuts are short bits of text which are expanded by pressing TAB; they are listed in the Code Snippets Manager. For example, if you have a snippet with the shortcut **ifelse** you could enter the text **ifelse** into the code editor and press TAB to auto-expand the snippet. **Note:** in order to enable TAB-expansion of snippets, '*Expand snippets by tab character*' must be enabled in the *Text Editor, Ruby, IntelliSense* page of the *Options* dialog.

### ADDING CUSTOM DATA TO A SNIPPET

When inserted into your code, some snippets have one or more highlighted fields ('replacement points'). If you hover your mouse pointer over a replacement point, a descriptive tooltip appears. You may edit a replacement point (say by changing it to the name of a particular variable) and, if there are more replacement points, you may move directly from one to the next by pressing the TAB key. Where multiple replacement points have the same name, editing one will automatically change the others to match.

## EXPANSION AND SURROUNDWITH SNIPPETS

Most snippets are 'expansion snippets' which means that their text is simply inserted into the code editor; if any other text happens to be selected at the time, it will be replaced by the expanded snippet. Some snippets are also defined to be 'SurroundsWith' snippets. If you select text in the code editor and then insert a SurroundsWith snippet, the snippet text will enclose the selected text. For example the **if** snippet is a SurroundsWith snippet. Let's assume that you have already entered the following into the code editor:

```
puts( 'hello world' );
```

You select this text, then right-click and choose **if** from the Snippet menu. When the **if** snippet is entered it automatically surrounds the selected text, resulting in this:

```
if true then
  puts( 'hello world' );
end
```

In this case, **'true'** is a replacement point and it is selected so that you can immediately replace it with your desired test condition.

## SNIPPETS IN THE COMPLETION LIST

You may optionally place snippets into the code completion list. This is the list which appears when you press **CTRL+Space**. The list will also appear when you enter text into the code editor if the *'Show completion list after a character is typed'* option is enabled. The completion list options are specified in the *Text Editor, Ruby, IntelliSense* page of the *Options* dialog.

## THE CODE SNIPPETS MANAGER

The Code Snippets Manager is selected from the *Tools* menu. To view the available Ruby snippets, select 'Ruby' in the Language combo and expand the Ruby folder in the left-hand pane. Select each snippet to view details such as its shortcut and type ('expansion' or 'SurroundsWith'). Click the help button (?) at the top right of the Code Snippets Manager to view its Visual Studio help entry.

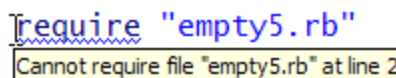
## Snippet Editor

You can create your own snippets using **Sniped** – a snippet editor for Ruby . **Sniped** is supplied with the Developer Edition of Ruby In Steel and comes with its own user guide. The snippet editor can be found in the `\Extras` subdirectory of your Ruby In Steel installation (typically located beneath `C:\Program Files\SapphireSteel Software\`).

## Wavy Lines – Syntax Error Indicators

Syntax errors and warnings are shown in the code editor by a wavy underline marking the location at which the error was detected. Note that the line appears only at the point at which the syntax becomes unambiguously incorrect. The actual position of the error may sometimes lie at a point in the code some way preceding the indicator. You may hover the mouse over a wavy line indicator in order to view an explanatory message in a tooltip.

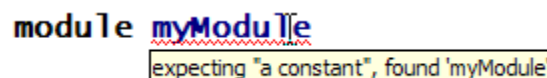
By default a blue wavy line indicates a warning:



```
require "empty5.rb"
```

Cannot require file "empty5.rb" at line 2

A red wavy indicates a syntax error:



```
module myModule
```

expecting "a constant", found 'myModule'

## Intelligent Type Inference

Ruby is a dynamic language. Among other things, this means that the types of variables are not predeclared and a single variable may have many different types during the lifetime of a program. This contrasts with languages such as C and Pascal in which the type of each variable is specifically declared and an attempt to assign any other type to that variable is an error.

While dynamic typing is useful for some programming tasks, it has the unfortunate side-effect of making it extremely difficult to provide accurate IntelliSense. Consider the problem: in C or Pascal, before you can use a variable **x**, you have to state its type – say an Array, an Integer or a String. Once the type has been declared, the IntelliSense system knows that whenever the variable **x** is encountered within a specific scope, it can treat it with certainty as a variable of the defined type. So if **x** is a String and the programmer enters **x** followed by a dot then the code-completion list will always display String methods.

Similarly the return type of functions are specifically declared in statically typed languages – so if the function **someFunction()** is declared to be an Integer, the IntelliSense system knows that it can always assume that the value returned by that function is an Integer.

With Ruby, no such assumptions can be made! Neither the variable **x**, nor the method, **someMethod()**, have predeclared types. It is quite possible for **x** to be an Array, a String and an Integer at various points, and within the same scope, during the execution of a program. The method **someMethod()** could, equally, have varying return types depending on how that method is called. In many cases, this makes it impossible (even in principle) for the IntelliSense system to work out exactly which type a variable is or which type a method returns – because the method and the variable vary their types at different times during the same program when it runs!

The Ruby In Steel IntelliSense system deals with this problem in two ways. In some cases, it is possible to infer the type of a variable at a given point in the code. For example, if a String such as “**Hello world**” is assigned to **x** or the method **someMethod()** explicitly returns an integer then Ruby In Steel will treat **x** as a string and **someMethod()** as an integer. If **x** is subsequently assigned some other type – say an Array – then Ruby In Steel will henceforward treat it as an array. These are relatively simple examples of type inference.

In fact, Ruby In Steel's intelligent Inference Engine not only infers the type of a variable within a given *scope*; it also infers its type within a given *context* – that is, it attempts to determine which type a variable will have at a certain point when the program is run. This is not foolproof, of course; Ruby In Steel analyses the code while it is still being written and the actual types of variables may not be certain until the program is run by the Ruby interpreter. Nevertheless, allowing for these constraints, Ruby In Steel constantly analyses your code in the background and attempts to determine the most likely types of each variable. So, for example, if `x` is a string on line 1, an Array on line 100 and an Integer on line 500, when you move to those lines to edit your code, Ruby In Steel attempts to provide the appropriate completion lists for the class of `x` at that given point in the code. The IntelliSense system calculates completion lists based on a variety of factors such as inheritance, visibility (public, private, class or instance), modularity and inclusion (mixed in modules).

There is currently no way, however, in which the parameters of a method can be reliably inferred. If you wish to have better Parameter Info IntelliSense for methods, you should consider using Ruby In Steel's Type Assertions.

## Type Assertions

Type Assertions are declarations of the expected types of parameters sent to a method and the return type of the method. These assertions are entirely optional. They are placed in a comment block immediately preceding a method and have no impact upon the meaning of the Ruby code itself. In other words, Type Assertions give you extra features when you are developing with Ruby In Steel but they have no effect on the code as far as the Ruby interpreter or any other editor or IDE is concerned.

When a method is preceded by type assertions, Ruby In Steel's Parameter Info tooltips are able to display information on the asserted parameter and return types, helping to resolve ambiguities and avoid accidental coding errors. Consider, for example, this method:

```
def addName( names, aName )  
  return names << aName  
end
```

It is, in principle, impossible to determine the types of the parameters, **names** and **aName**, or the class of the return value. Even the << is not a sufficient clue as this is defined for many different classes in Ruby. Consequently, the **addName()** method could legally be called in the following ways, in each case both the parameter and the return types being different:

```
addName( "Fred,Mary,", "Simon" )
addName( ["Fred","Mary"], "Simon" )
addName( ["Fred","Mary"], ["Simon"] )
addName( 123, 456 )
```

But let's suppose that the programmer who wrote the **addNames()** method had done so with the specific intention that the first parameter, **names**, should always be an array, the second parameter, **aName** should always be a string and the returned value should always be an array. The programmer could, of course, document this fact somewhere – but there is no guarantee that the person calling the method will ever read the documentation. By using Type Assertions you can bind your documentation to Ruby In Steel's IntelliSense.

## MAKING TYPE ASSERTIONS

Type assertions are entered into a comment block immediately above a method. The assertion may include a return type and one or more arguments matching the argument list.

The *syntax* for assertions can be summarized like this:

```
# :return: => <ReturnType>
# :arg: <ArgName> => <ArgType>
# <Optionally More Args...>
```

**Note:** When no return type is given, *NilClass* is assumed. The argument assertions should be entered in the same order as the actual parameters. The names of arguments used in the assertions will be used by IntelliSense even if those names differ from those used in the actual parameter list. If more arguments are asserted than actually exist, the superfluous assertions are ignored. If fewer arguments are asserted than actually exist, the extra arguments will default to *Object* when there is a return value and to *NilClass* when there is no return value..



Example:

```
#:return:=>Array
#:arg:names=>Array
#:arg:aName=>String
def addName( names, aName )
  return names << aName
end
```

Given the assertions above, the parameter info will now show the following:

```
addName(|
addName (names:Array, aName:String): Array
```

## Automating Type Assertions

You can add a type assertion comment block over a method by entering two # characters on a blank line above the method. For example, let's assume you have this method:

```
def addName( names, aName )
  return names << aName
end
```

If you enter ## onto the line above this method, Ruby In Steel will automatically insert the following comment block:

```
#:return: => Object
#:arg: names => Object
#:arg: aName => Object
```

You may now edit this to show the desired types. For example:

```
#:return: => Array
#:arg: names => Array
#:arg: aName => String
```

Those types will now be displayed by IntelliSense.

## The IntelliSense Librarian

The IntelliSense Librarian is an optional tool which pre-compiles selected Ruby files and makes their IntelliSense information available to your project. This has three main benefits:

- **It provides enhanced Rails IntelliSense including 'database IntelliSense'**

Enhanced IntelliSense is automatically provided for Ruby On Rails projects. This includes code completion and other IntelliSense features such as documentation tooltips for ActiveRecord, ActionController, ActionView and related classes and methods. You may also generate application-specific database IntelliSense based on the database schema.

- **It makes additional IntelliSense available to Rails and other web frameworks**

When using a frameworks such as Ruby On Rails it is frequently possible to refer to classes and methods even though the source files in which those classes and methods are defined are not specifically 'required' in your code. This is because the framework references the necessary files at runtime. The Ruby In Steel IntelliSense engine - in common with the Ruby interpreter itself - only works with complete 'runnable' Ruby programs. Normally this means that IntelliSense is only provided for classes in the current file or any classes that are required or included in accordance with Ruby syntax. However, you can bypass this limitation by creating precompiled libraries of any additional files required by Rails or another framework. When you add the library files to the Solution Explorer, your code will have access to all the relevant IntelliSense information just as though they had been 'required' explicitly in your Ruby code.

- **It increases the speed of parsing large source code libraries**

Some Ruby libraries (for example, **date** or **yaml**) are very large and complex and some libraries may also 'require' many other Ruby files. As the Ruby In Steel IntelliSense engine interprets and analyses source code in order to provide accurate code completion, the analysis of very complex source code files may have an adverse effect upon the speed of code completion. In most cases, you will not alter the code in these required libraries, so it would be more efficient if the IntelliSense engine did not constantly reanalyze the source code. When you compile the code into a library, the IntelliSense engine will obtain information from the library instead of analyzing the Ruby source code, resulting in a speed improvement.

**NOTE:** Many Ruby programmers may never need to use the IntelliSense Librarian. For most ordinary Ruby programs, our automatic IntelliSense engine will provide the full range of code completion features almost instantly. You only need to use the Librarian if you need the additional capabilities described above.

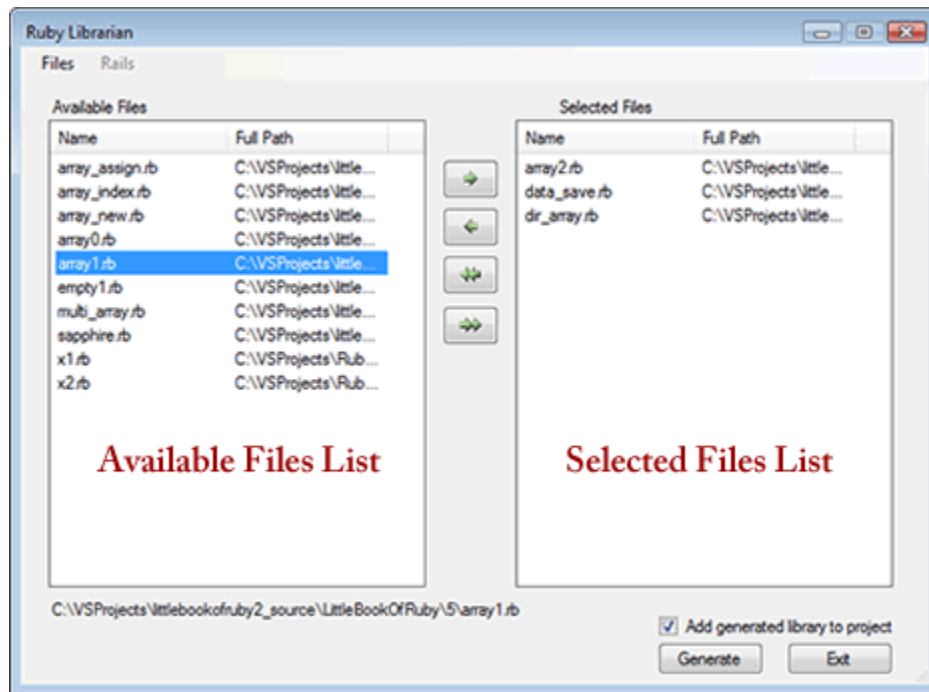
## CREATING A NEW INTELLISENSE LIBRARY

The following provides a simple example of using the Librarian to generate an IntelliSense Library from some Ruby source files and automatically add that Library to the currently loaded project.

- Click the Librarian menu item on the Ruby menu or Ruby toolbar.
- This loads the Librarian.
- Click Files/Add Files.
- In the Open dialog, select one or more source code files and click 'Open'.
- Your selected files will now be added to the 'Available Files' list.
- Click the 'Add All' (right-facing double arrow) button to add all available files to the 'Selected Files' list.
- Ensure the 'Add generated library to project' option is checked.
- Click the Generate button.
- When prompted, enter a file name, without an extension - such as *mylib*. Then click 'Save'.
- The compiler will now generate an IntelliSense library based on the selected source files. After a short while (depending on the size and complexity of the library), a dialog will inform you that the Library generation is complete. Click 'OK'.
- Click 'Exit' to close the Librarian. You will now see your Library beneath the Solution Explorer 'References' node in your Project.
- The IntelliSense information is now available for use in your project even to those source files which do not explicitly 'require' the original Ruby code files.

## Librarian Reference

Once you have started the Librarian by clicking the Librarian icon you will see this dialog:



*The IntelliSense Librarian*

### THE FILES MENU

The Files menu has the following items...

#### ➤ ADD DIRECTORY

This opens a dialog from which you may select a directory. All the Ruby source files in the selected directory will be added to the IntelliSense library. Subdirectories are not added. However, files that are 'required' in the source files will be added even if the required files are in different directories. Click 'OK' to save all the source files from the selected directory into the Librarian's 'Available Files' list.

#### ➤ ADD FILES

This opens a dialog from which you may select one or more Ruby source files. In this dialog, you may press Shift when clicking the mouse in order to select multiple files. Click 'Open' to save the selected files into the Librarian's 'Available Files' list.

#### ➤ CLEAR

This deletes all files from the 'Available Files' list. This has no effect on the original files stored on disk.

## THE RAILS MENU

This menu is only available when the current project is a Rails project. This should be set by default for rails projects and can also be selectively enabled by *Project/Properties*. The Rails menu has the following items...

### ➤ BUILD RAILS LIBRARY

Ruby In Steel auto-generates a pre-compiled Rails IntelliSense library and *you should not normally need to use this option* unless you install a significantly new version of Rails which requires additional IntelliSense. When you use this, you may need to browse to the directories containing the relevant Rails source code. The Rails directories are normally located beneath your Ruby\gems directory.

For example: `c:\ruby\lib\ruby\gems\1.8\gems\`

You may need to browse to specific directories.

For example: `c:\ruby\lib\ruby\gems\1.8\gems\activerecord-2.0.2\lib\active_record\`

The relevant Rails directories are shown below - here, we substitute `<Ruby\Gems>` for the actual Ruby\gems path on your system. The version numbers of the directories (here 2.0.2) will vary according to the installed version of Rails:

<b>Active Record:</b>	<code>&lt;Ruby\Gems&gt; \activerecord-2.0.2\lib\active_record\</code>
<b>Action Controller:</b>	<code>&lt;Ruby\Gems&gt; \actionpack-2.0.2\lib\action_controller\</code>
<b>Action View:</b>	<code>&lt;Ruby\Gems&gt; \actionpack-2.0.2\lib\action_view\helpers\</code>
<b>Unit Test:</b>	<code>&lt;Ruby\Gems&gt; \activerecord-1.15.3\lib\active_record\</code>

You may select or deselect Rails generator options by clicking the check boxes alongside each.

**Please note:** Rails IntelliSense generation is a very time-consuming process which may take 15 minutes or longer. Visual Studio will not be available for use during this time. Please wait. *Do not shut down Visual Studio until the Rails IntelliSense generation has completed.* A progress bar shown at the bottom of the Visual Studio environment indicates the approximate time to completion. At the end of the process you will see a dialog informing you that the Rails IntelliSense Library was built successfully.

## ➤ DATABASE INTELLISENSE

Database IntelliSense is designed to provide code completion within Rails views (inside embedded Ruby templates such as *new.html.erb*) and is generated from the database schema files which are created when you perform Rails migrations. A schema file is normally found in the Rails `\db` directory. Note that if your schema is not visible, you may need to synchronize the Solution Explorer (*Ruby/Synchronize*).

### TO CREATE DATABASE INTELLISENSE:

- Select the schema file (normally *schema.rb*) when prompted and click 'Open'.
- Click 'OK' when asked to confirm that you wish to build the database schema.

## THE BUTTONS

- **RIGHT ARROW**  
Add highlighted file in the Available Files list to the Selected files list.
- **LEFT ARROW**  
Remove the highlighted in the Selected Files list file from the Selected files list.
- **DOUBLE-RIGHT ARROW**  
Add all files in the Available Files list to the Selected files list.
- **DOUBLE-LEFT ARROW**  
Remove all files from the Selected files list.
  
- **GENERATE**

This compiles an IntelliSense library (a 'symbol table') using information from the files in the Selected Files list. When you select 'Generate' a dialog will prompt you for a file name. You should enter a descriptive file name (without an extension) such as '*mynumericlib*'. The extension '*.rst*' (Ruby Symbol Table) will be added automatically. You may save the library into any directory but we recommend using the default '*\SymbolTables*' subdirectory which is stored beneath your *\AppData\Local* folder.

**Note:** If you wish to add the generated library to the current project (this is the default), make sure that the 'Add generated library' option is selected.

### Example:

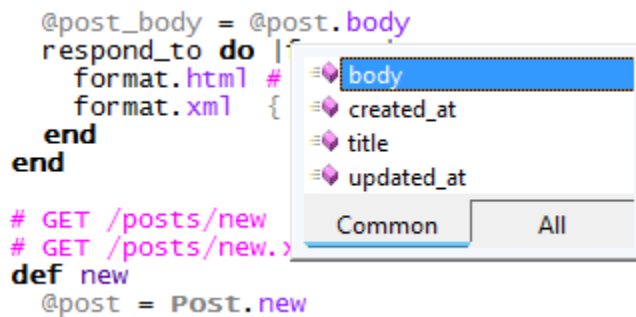
Let's assume that your *schema.rb* contains this code:

```
create_table "posts", :force => true do |t|
  t.string "title"
  t.text "body"
  t.datetime "created_at"
  t.datetime "updated_at"
end
```

Now in your Posts controller, (*posts\_controller.rb*), let's assume you have the following method:

```
def show
  @post = Post.find(params[:id])
  # etcetera...
end
```

With database IntelliSense you can now enter a dot after the variable, **@post**, and you will see methods defined for the database columns listed in the 'Common' page. These database methods will also be available in the 'All' page along with many other methods for a Post object...



Database IntelliSense is also available for the instance variable matching the controller name (e.g. **@post**) when you are writing code in a view (e.g. *\posts\show.html.rb*) between embedded Ruby tags (<% and %> or <%= and %>).

### ➤ EXIT

The Exit button closes the Librarian.

## To Add Libraries to A Project

If you have previously created a Library file (a 'Ruby Symbol Table' with the extension *.rst*) which you wish to use in the current project, follow these steps:

- In The Solution Explorer, right-click the *References* node and select 'Add Reference'.
- Select the *Browse* tab and either select one of the listed *.rst* files or, if necessary, browse to the directory containing it (the default is *\AppData\Local\SapphireSteel Software\SymbolTables*).
- Select the library (*.rst*) file(s) and click the *Add* button.
- When all the desired library files have been added, click *OK*.
- The libraries should now be listed beneath the *References* branch of the Solution Explorer.



## Code Navigation

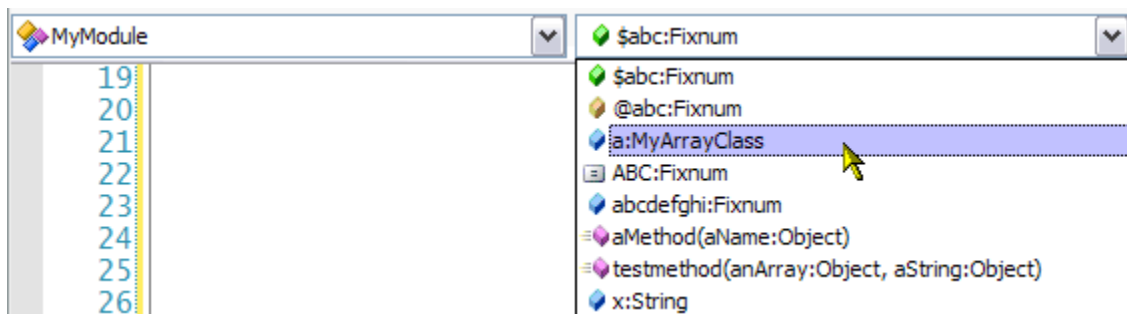
### [ Some Features Developer Edition only ]

Ruby In Steel has a variety of tools to help you find the code you are looking for.

- > *Navigation Bars* let you locate classes and methods in the current document
- > *Go To Definition* helps you find the definition of variables, methods and classes
- > *Bookmarks* can mark places in code so that you can navigate back to them quickly
- > *Find and Replace* tools let you search through one file or an entire project
- > *The Ruby Explorer* locates code and documentation of classes and methods

### Navigation Bars

At the top of the code editor window you will see a navigation bar containing two drop-down combo boxes.



The box on the left-hand side lists any classes and modules in the current code file. The box to its right lists methods, constants and variables. You can select an item from either of these combo boxes in order to locate its definition in the code editor.

### Go To Definition

Right-click the name of a method, class or variable in your code and select *Go To Definition* from a popup menu in order to locate the definition of the selected item. If the definition is in a required file or a Ruby Class Library file, the file will be automatically loaded into the editor. If the definition is in one of the C-language files of the Ruby class library, a Ruby documentation file containing the embedded documentation and empty Ruby-syntax declarations ('stubs') of the original C-language classes and libraries will be loaded.

## Bookmarks

You can place bookmarks at specific locations in your code in order to be able to return to those locations quickly. Bookmarks can be managed using the Bookmarks Window (available from the *Views, Other Windows* menu) or from the Text Editor Toolbar. Bookmarks are documented in more detail in the main Visual Studio help system.

## Find and Replace

Ruby In Steel supports all the default Visual Studio Search tools to let you find and replace text in the current document or in multiple files. Optionally you can search with wildcards and regular expressions. Searching tools are available from the *Edit, Find and Replace* submenu.

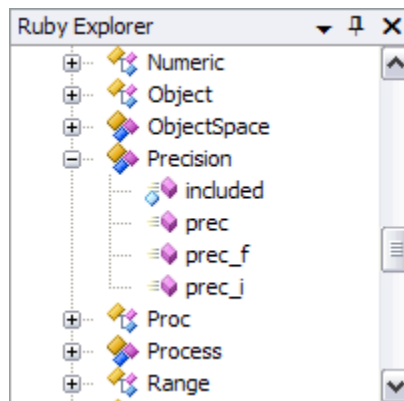
## Incremental Search

You can search incrementally in the current document by entering a sequence of characters. A highlight will move through the code to find the first occurrence of matching text (if any). Incremental Searching is available on the *Advanced, Edit* submenu (shortcut: **CTRL+I**).

## Go To Line

You can move your cursor to a specific line number using the *Go To* dialog (shortcut: **CTRL+G**) from the Edit menu.

## The Ruby Explorer



The Ruby Explorer is a class browsing tool which can be used to view an alphabetical outline of Ruby classes and methods. The Explorer shows the classes of the standard Ruby library plus the classes defined in the current project. The methods of each class can be viewed by clicking the + symbol to the left of the class name.

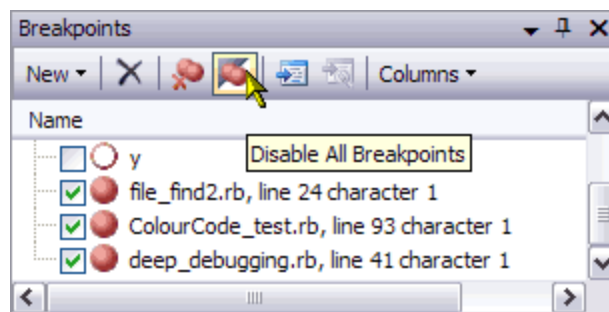
You can double-click a class or method name in order to navigate to its source code (if available) and any embedded documentation. Much of Ruby's class Library is written in the C language; in these cases, the Ruby Explorer will load up a Ruby documentation file which has been generated from the original C file and it will navigate to the documentation relating to the selected class or method.

## The Debugger

RUBY IN STEEL INCLUDES INTEGRATED DEBUGGING TO HELP TO FIND AND FIX ERRORS IN RUBY AND RAILS APPLICATIONS. THESE ARE THE ESSENTIAL FEATURES OF THE DEBUGGER...

Ruby In Steel Developer uses our fast '*Cylon*' Debugger by default; the slower Ruby Debugger is provided with the Text Edition. Not all debugging operations are available in the Text Edition Debugger.

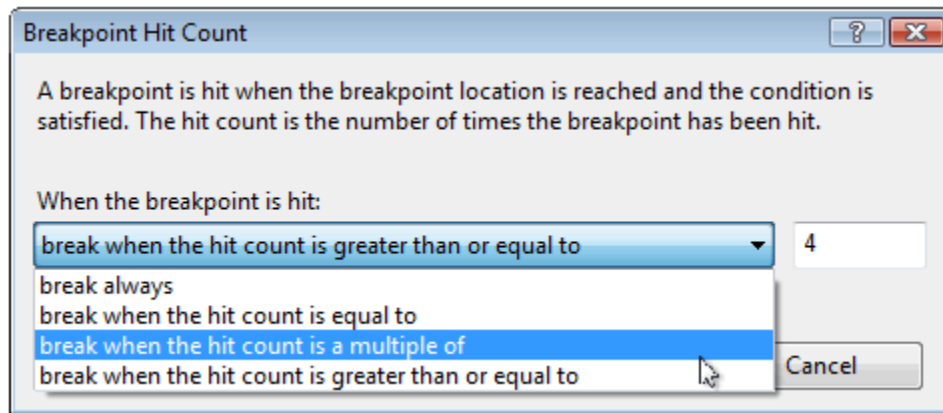
### Breakpoints



Ruby In Steel lets you add breakpoints to your Ruby programs by clicking in the margin of selected code lines. You can selectively enable/disable selected (or all) breakpoints in this breakpoints window. You can also remove an existing breakpoint by clicking it in the margin of the code editor.

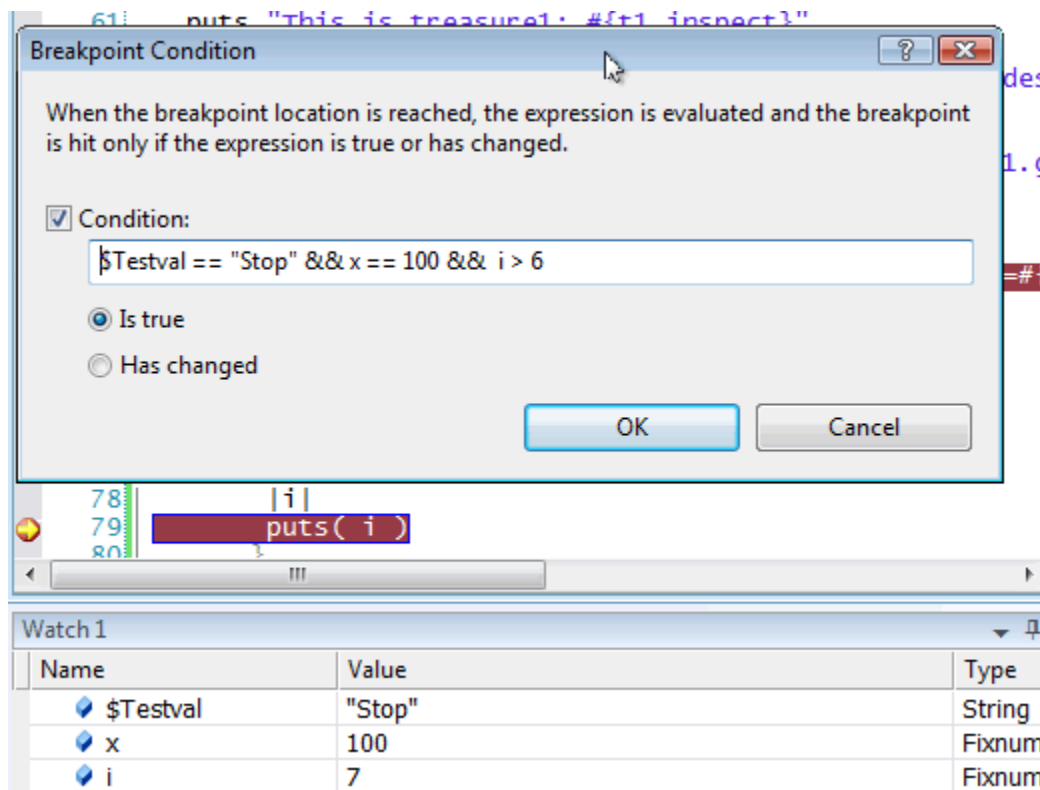
**Note:** Once a breakpoint has been added, you may right-click it (in the margin) in order to set a number of conditions as explained in the following pages...

## Break On Hitcount



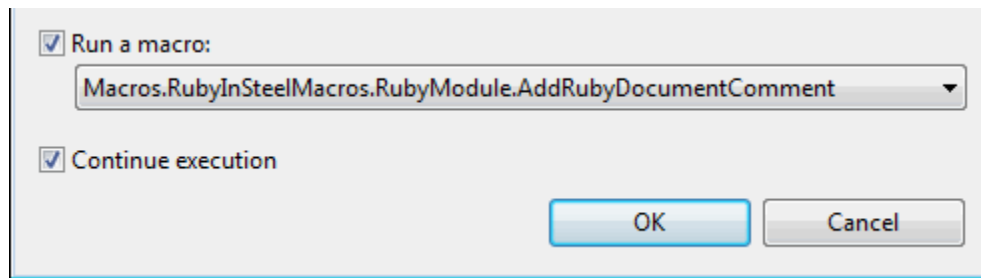
To break after a specific piece of code has executed a certain number of times, add a *Breakpoint Hit counter*. Do this by right-clicking the breakpoint and selecting a condition such as *'break when hitcount is multiple of'* from the drop-down list. Add the integer value to test.

## Conditional Breakpoints



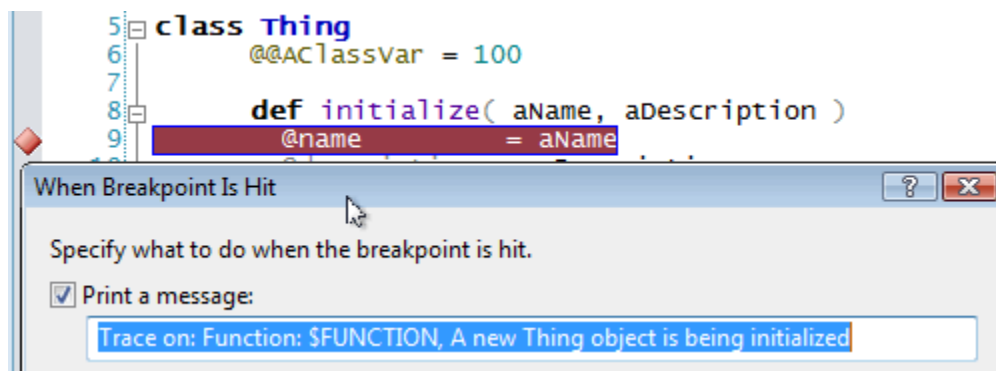
If you only want to break when a condition (a test of one or more values) is met, create a *Conditional Breakpoint*. This can be entered in normal Ruby syntax.

## Run Macro On Break



You can attach a macro to a Breakpoint - for example, you could write or record a macro to do anything from popping up a dialog box to inserting comments into your code - and this will run when the breakpoint is hit.

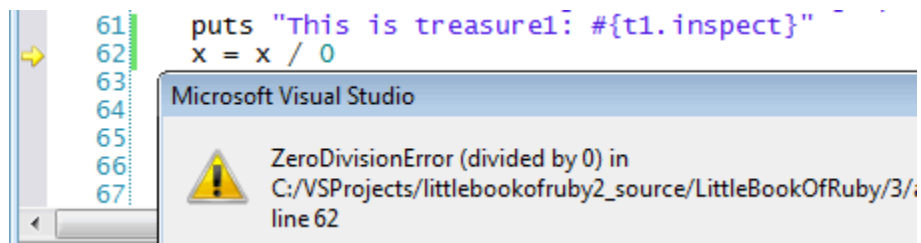
## Tracepoints



If you wish to be alerted that a piece of code has executed but you don't want execution to break you should add a Tracepoint. When a Tracepoint is hit it will print the specified message to the Output window without causing the program to stop.

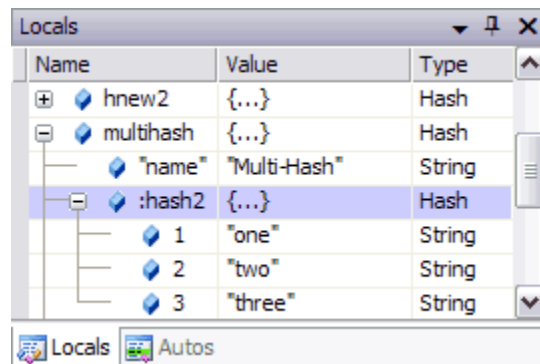
Tracepoints are set by right-clicking a breakpoint and selecting the 'Print a message' option in the dialog. You may place the names of variables between curly brackets ( for example, @xxx) in order to display their current values. The keyword **\$FUNCTION** displays the name of the currently executing method.

## Break On Exception



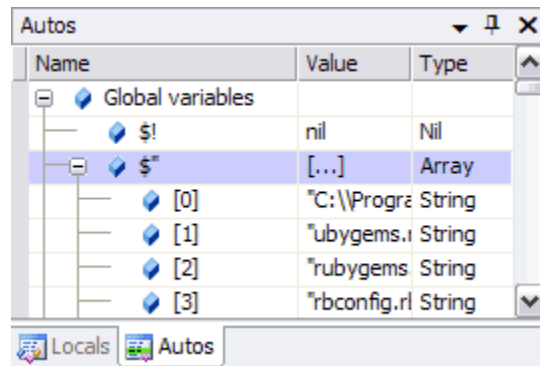
If Just-in-time Debugging is enabled (*Tools, Options, Projects and Solutions, Ruby In Steel*), a debugging session will break when an exception occurs. When this happens you can examine the state of the program using the usual range of debugging windows.

## Locals window



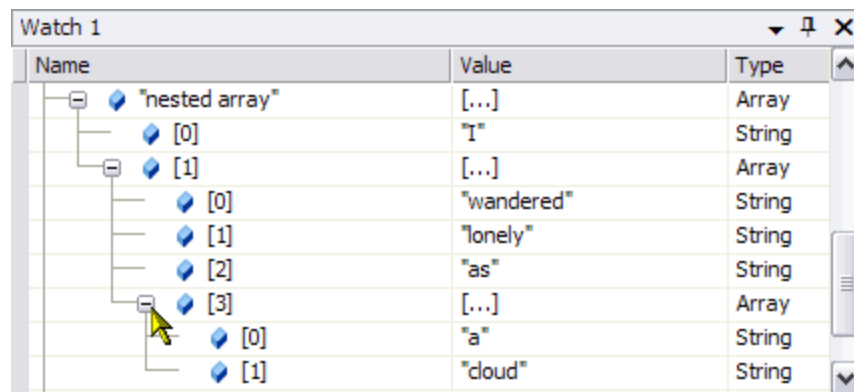
The Locals window automatically monitors and displays the values and types of local variables and you can 'open up' items such as arrays and hashes in order to drill-down into variables.

## Autos Window



The Autos Window optionally displays the values and types of global, class and instance variables, object methods and singleton methods available, plus **self**. The information displayed may be configured globally for all new projects (*Tools, Options, Projects and Settings, Ruby In Steel*) or for just the current project (*Project, Properties*). Note that some projects may require the evaluation of large amounts of data - particularly for **self**. If debugging seems slow, you may want to experiment with the settings for this window.

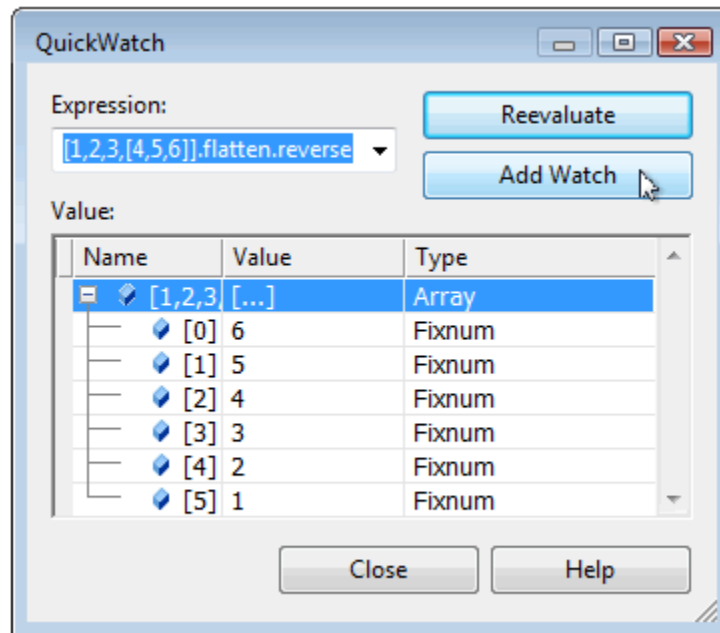
## Watch Window



You can enter the names of variables (or drag and drop variables from the editor window) in the Watch window. Ruby In Steel Developer lets you 'drill-down' into watch variables to view finer levels of detail. For example, you can open up Arrays and Hashes to view the objects they contain. To drill-down, click the + symbol to the left of each variable to open up sub-branches of information.



## Quick Watch Window

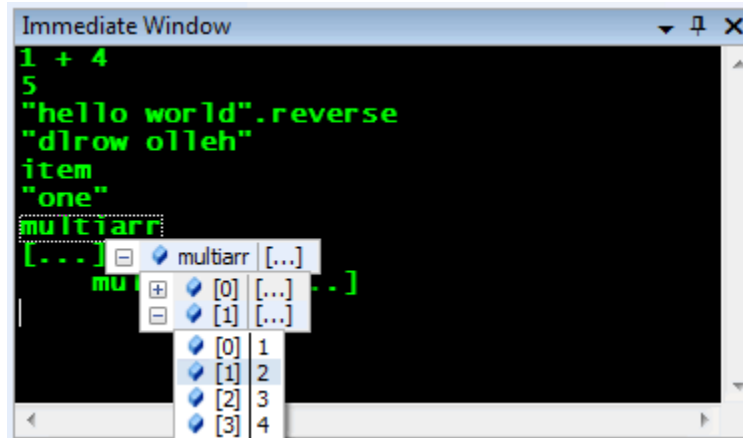


The Quick watch window (available on the main *Debug* menu) is a convenient place for trying out expressions. Here, for example, you might ask Ruby to evaluate an expression such as the following:

```
[1,2,3,[4,5,6]].flatten.reverse
```

The result (the two arrays merged into one and reversed) can be inspected in a 'drill down' Quick Watch viewer. If you want to place a permanent watch on a variable or expression, click the 'Add Watch' button to place it into a docked Watch window.

## Immediate Window



The Immediate Windows is useful for trying out bits of code when stopped at a breakpoint. Here you can evaluate simple expressions such as:

```
1 + 2
```

And you can enter assignments such as:

```
myarray = [1,2,3,[4,5,6,[7,8,9]]]
```

You can also evaluate expressions such as:

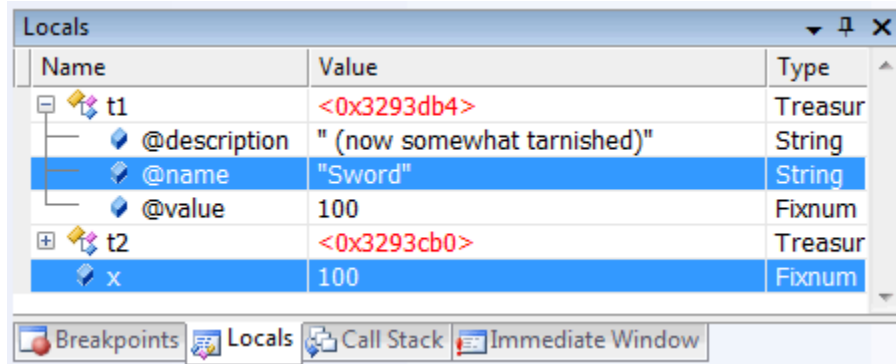
```
newarray = myarray.reverse
```

The Immediate window has no convenient way of displaying complex data structures, so an array, for example, will be shown simply as `[..]`. However, when you evaluate local variables these will automatically be shown in the Locals window. Alternatively you can hover over a variable name in the Immediate window in order to drill down into it.

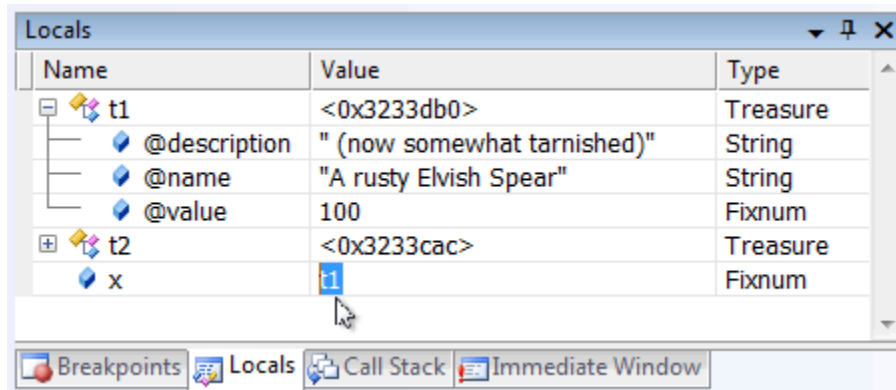
The Immediate window includes the ability to copy and paste text and you can press the up and down arrow keys to scroll back and forth through the sequence of previously entered expressions.

## Dynamic Debugging

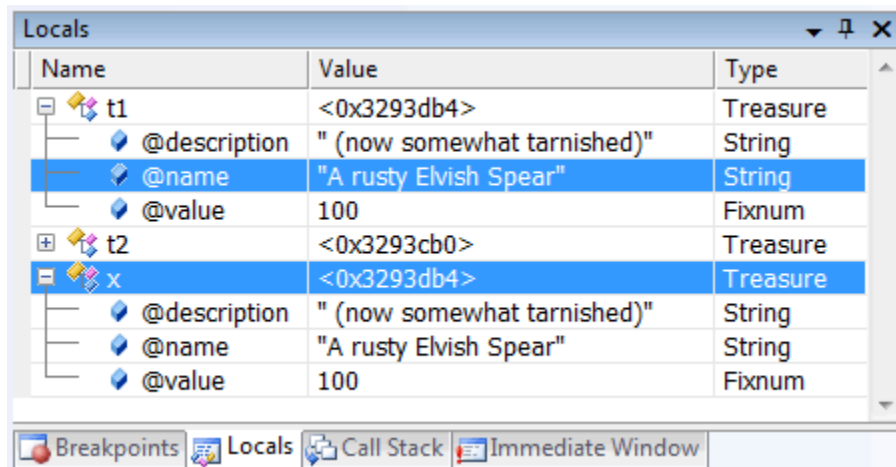
When stopped at a breakpoint *Cylon* lets you edit the values of variables in debug windows such as Watch and Locals. Select a variable and click to put it into Edit mode...



You can even change the type of a variable in this way. Here the Fixnum, **x**, is selected and the variable is now being assigning a Treasure object, **t1**...

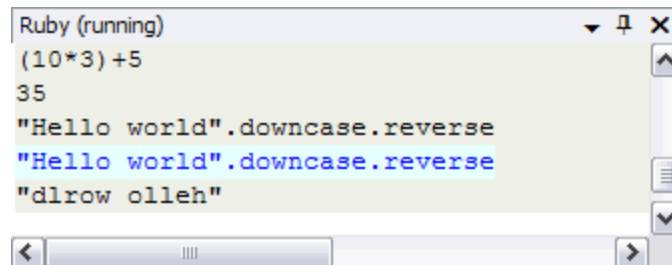


Now, **x** has been assigned **t1** and its class type is now Treasure rather than Fixnum...

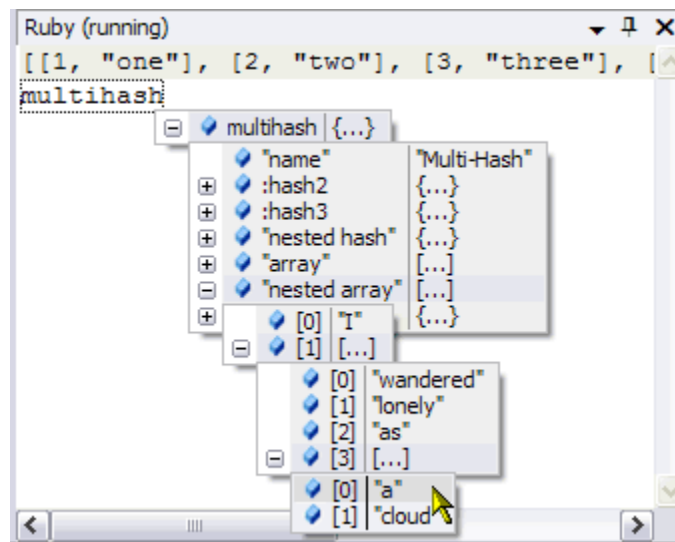


## Evaluate Expressions In The Ruby Console

When stopped at a breakpoint, you may evaluate expressions or you may call Ruby methods in the Ruby Console. To evaluate a variable, simply enter its name.

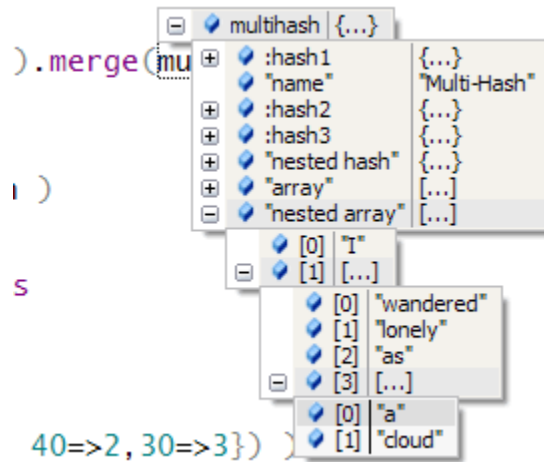


## Hover and Drill-Down In The Ruby Console



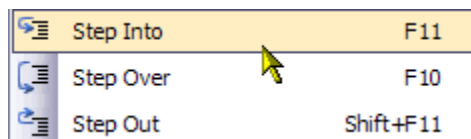
When you have stopped at a breakpoint, you can enter the name of a variable into the editable part of the integrated Ruby console. Hover the mouse over the variable name to view drill-down details of that variable.

## Hover and Drill-Down in the Ruby Editor



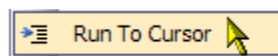
When paused at a breakpoint, you can hover the mouse over a variable name in the code editor in order to display drill-down details of that variable.

## Tracing with Step Into / Step Over / Step Out



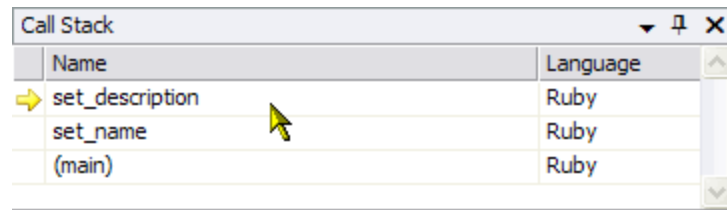
When you hit a breakpoint, press F10 to continue tracing through the current block of code (Step Over) or F11 to trace into any methods called by the current code (Step Into) or SHIFT+F11 (Step Out) to step out of the current method and continue from the next line of the calling routine. These commands are also available on the *Debug* menu.

## Run To Cursor



Right-click a line of source code and select 'Run To Cursor' from the popup menu. Your program will pause on the selected line and you can then use the debugging features (trace, watch etc.)

## Call Stack



When your code is running several methods deep you can see this in the Call Stack pane. Double-click items to navigate through the stack. (**Call stack navigation available in Developer Edition only**).

### HOW TO USE THE CALL STACK

In effect, the Call Stack lets you ‘trace backwards’ through the execution of your Ruby programs. Each time you step into a new method with the debugger, a new entry is added to the call stack. So, if you enter method **x**, the call stack shows **x** along with (optionally) some other information such as the name of the source file and the line number. (Right click the Call Stack window and select options from a popup menu to configure the visible details). If some code in the **x** method calls the **y** method, the call stack will show entries for **x** and for **y**. These are not just static indicators of the methods which your program has passed through; they are active ‘moments’ in the flow of execution which can be recalled just by clicking each entry in the call stack.

So if, for example, you have a parameter called **aString** which has the value “hello world” when it is passed to **x**; has been changed to “HELLO WORLD” by the time it is passed to **y** and has become “DLROW OLLEH” when it arrives at **z**, you can flip up and down the call stack, recalling each point of execution and view the changing values of **aString** at each point.

**Tip:** View the changing values of variables in the Watch or Local window as you navigate the Call Stack.

This even works with recursive methods. If a method **calc** increments a variable **sum** as it repeatedly calls itself, you can navigate up and down each separate instant at which the **calc** method was recursively called to check on the changing value of **sum**.

### STOP DEBUGGING

Press SHIFT+F5 to stop debugging.

## Run, Debug, Build

### Run

#### A) IN THE RUBY CONSOLE

To run a program without debugging inside the Visual Studio environment...

Press **CTRL+F5** or **CTRL+2**.

Or select *Ruby* menu, *Ruby Run*.

Or select *Debug* menu, *Start without debugging*.

You can now interact with the program within the Ruby Console (if this is not visible, select the *View* menu, then *Other Windows* and *Ruby Console*).

The Ruby Console may either be placed in its own tabbed page or it may be docked or 'floated'. You can change its tabbed, docked or floating behavior by right-clicking its caption bar and making a selection from a popup menu.

#### B) IN A COMMAND WINDOW

To run a program without debugging inside a popup 'command window'...

Press **CTRL+1**.

Or select *Ruby* menu, *Ruby command*.

### Debug

To run a program with debugging... press **F5**

Or select *Debug* menu, *Start Debugging*.

### Build To Check For Syntax Errors

**Build Solution** and **Rebuild Solution** examine the files in a project and report on any syntax errors. Unlike the Build option for compiled languages, Ruby In Steel's *Build/Rebuild* does not create an executable file.

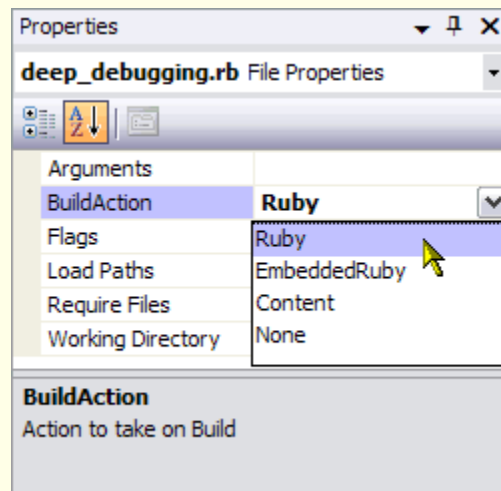
To **Build** a Solution, select *Build, Build Solution*.

To **Rebuild** a Solution, select *Build, Rebuild Solution*.

You can view the progress of the error checking in the Output window. Errors are reported in the Error List. **Build** only checks those files which failed a previous Build or have been changed since a previous Build; **Rebuild** checks *all* files.

As part of the Build process, Steel creates a `\SyntaxCheck` directory containing a record of all the documents built (these are 0-length files). If you wish, you can remove this directory by selecting *Clean Solution* from the *Build* menu. Once this is done, the next time you perform a Build, all the files in the solution will be checked (that is, Build will perform the same action as Rebuild).

### Setting The BuildAction Property



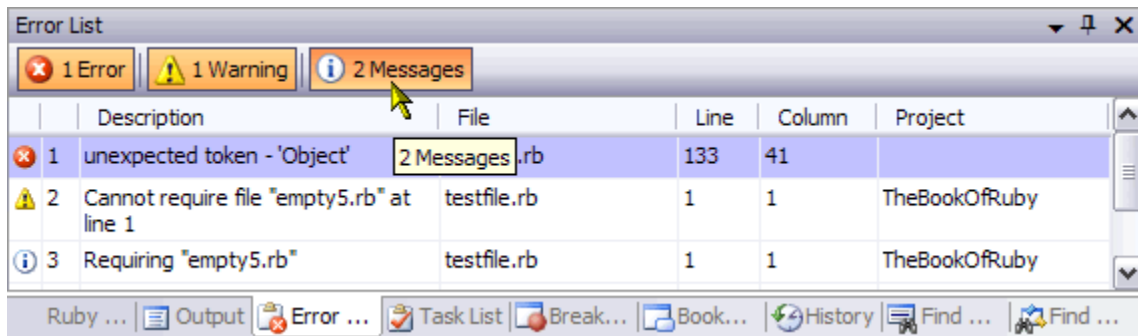
Build and Rebuild only check those files with a BuildAction property. The BuildAction of Ruby (.rb) files is, by default, **Ruby**; the BuildAction of Rails (.rhtml) files defaults to **EmbeddedRuby**. If you want to exclude a file from the syntax checking performed by Build, select that file in the Solution Explorer and, in the Properties window, change its BuildAction property to **None**.



## Compile

The *Compile* option on the *Build* menu (CTRL+F7) is used to recompile the IntelliSense database. If, for any reason, the IntelliSense is not up to date with your changes, you may select this to force a re-parse of IntelliSense data. **Note:** *IntelliSense requires syntactically correct code. If any syntax errors are flagged, you should correct these before selecting the Compile option.*

## Error List



The Error List window shows message on a separate row; it displays the name of the file in which the problem occurs and the line on which Ruby believes a syntax error to be found. Click the item to locate the problem line of code.

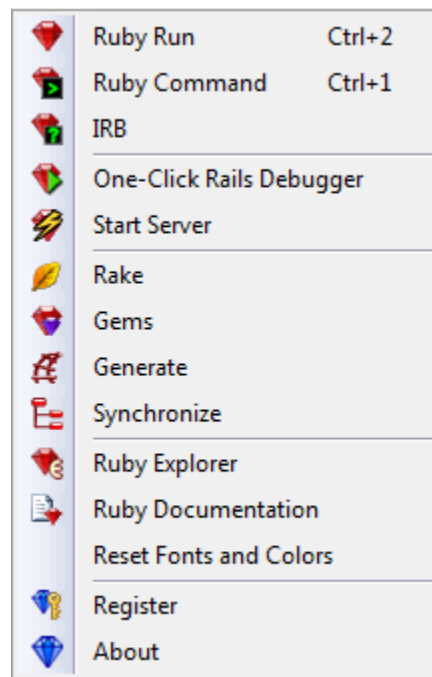
There are three types of message which may be displayed in this window: Error, Warning and Message. You can toggle the display of these messages by selecting the appropriate tabs. In the code editor, the location of an error is indicated by a red wavy line; the location of a warning is indicated by a blue wavy line.

## Ruby and Rails Tools

A SET OF TOOLS AND DIALOGS IS AVAILABLE ON THE RUBY MENU AND TOOLBAR. THIS IS A GUIDE TO THEIR PRINCIPAL FEATURES...

### The Ruby Toolbar and Menu (some items only in Developer Edition)

The Ruby menu and Toolbar provide access to a number of tools for use with Ruby and Rails.



*The Ruby Menu*



*The Ruby Toolbar*

### THE RUBY TOOLS

- > **Ruby Run** – runs the currently active file in a docked window
- > **Ruby Command** - runs the currently active file in a system ('command') window
- > **IRB** – runs the Interactive Ruby shell
- > **One-Click Rails Debugger** – starts debugging a Rails project
- > **Start Server** – starts a server (for Rails)
- > **Rake** – displays a toolwindow to run Rake
- > **Gems** – loads a dialog to install a Gem package
- > **Generate** – displays a toolwindow to run Rails generate scripts

- > **Synchronize** – synchronizes the Solution Explorer with files on disk
- > **Ruby Explorer** – loads the Ruby code and documentation explorer
- > **Ruby Documentation** - displays RDOC window
- > **Reset Fonts and Colors** - restores font and color defaults
- > **Register** - displays dialog to register this copy of Ruby In Steel
- > **About** – displays version and registration details of Ruby In Steel

More information on *Ruby Run* and *Ruby Command* can be found in the chapter '[Run, Debug, Build](#)'; the *One-Click Debugger* is described the chapter, '[Rails Development](#)'; the *Ruby Explorer* is described in the chapter, '[Code Navigation](#)'. The remaining tools are described below.

## RUBY TOOLBAR

The Ruby Toolbar can be displayed by selecting it from the *View/Toolbars* menu. Some items (such *Start Server* and *One-click Rails Debugger*) are only displayed when the current project is a Rails Project.

## RUBY MENU

The items on the Ruby menu duplicate the items on the Ruby Toolbar. As with the Ruby Toolbar, some items are only displayed when the current project is a Rails Project.

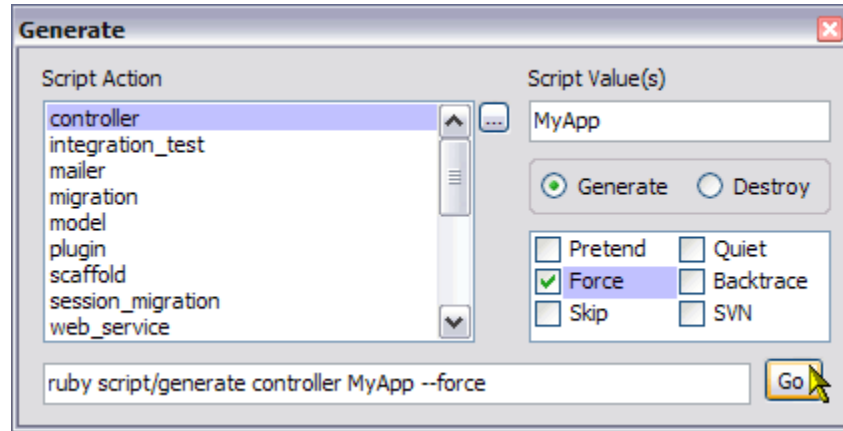
**Note:** If you wish to enable additional menu and toolbar items in a non-Rails project, set the *Rails/Rails Project* property to **true** in *Project/Properties*. You will need to close and reopen the solution for this change to take effect.

## IRB – The Interactive Ruby Shell

IRB is a Ruby tool which lets you interact with a Ruby interpreter at a command prompt. Click the IRB icon to run IRB.

## Generate (Rails)

[Developer Edition Only]



Use the **Generate** window to run Rails scripts without having to go into a command prompt.

### SCRIPT ACTION (GENERATOR)

The script action - also known as a 'generator' - is the name of the script you wish to run. A list of script actions is provided as standard and this list can be modified by the user. The default list includes commonly used scripts such as *controller*, *model*, *scaffold* and *migration*.

### SCRIPT VALUE

The script value is the name of the item you wish to create (or destroy). For example, if you wish to generate a controller named *MyBlog*, you would enter *MyBlog* into the Value field.

### GENERATE/DESTROY

You should select either the *Generate* or *Destroy* radio button depending on whether you wish to create (generate) or remove (destroy) an item.

## OPTIONS

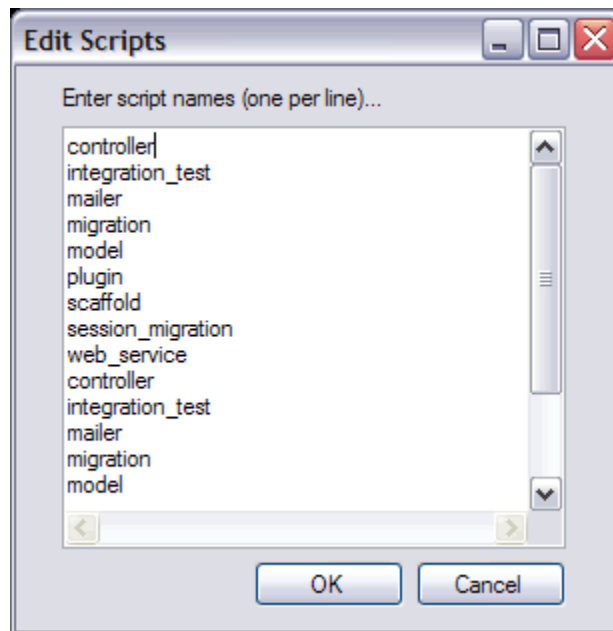
The following options are provided, shown here with their script equivalents and a brief explanation of their effects:

Option	Script Arg	
<b>pretend</b>	<code>--pretend</code>	Run but do not make any changes.
<b>force</b>	<code>--force</code>	Overwrite files that already exist.
<b>skip</b>	<code>--skip</code>	Skip files that already exist.
<b>quiet</b>	<code>--quiet</code>	Suppress normal output.
<b>backtrace</b>	<code>--backtrace</code>	Debugging: show backtrace on errors.
<b>svn</b>	<code>--svn</code>	Modify files with subversion. ( <b>Note:</b> <i>svn</i> must be in path)

## RUNNING/EDITING THE SCRIPT

As you make selections and enter values, the generator script will be automatically entered into the text field at the bottom of the window. If you wish to edit this, you may do so. When the script is complete, you may run it by pressing the 'Go' button. This will generate (or destroy) the selected item(s). The Solution Explorer will be updated to show the changes.

## CUSTOMIZING THE SCRIPT ACTION LIST



To change the available script actions, click the [...] button at the top right of the Script Action list and enter new actions, one per line, or delete existing actions. Click OK to confirm your edits. The edited Script Action list will now be used as the default.

## PLACING THE GENERATOR WINDOW

If you need to use the Generator window regularly, you may wish to dock it in the Ruby In Steel environment. Right-click its caption bar, select 'Dockable' and drag it to place it in a dock site. If you use it irregularly or you wish to place it on a non-primary monitor, right-click and set to 'Floating'. Click the caption bar 'Close' button when you have finished using the window. You can also use the window as a tabbed document page.

## Start Server

When developing Ruby On Rails applications you must start a web server in order to run or debug.

- You do *not* normally need to click *Start Server* when using **WEBrick** or **Mongrel**.
- You *do* need to click **Start Server** when using LightTPD.

## MORE INFORMATION

Ruby In Steel provides built-in support for the WEBrick, Mongrel and LightTPD servers. The *Start Server* button runs the currently selected server script. When using LightTPD, you must run the server explicitly by clicking the *Start Server* button. When using Mongrel or WEBrick, the server will be started automatically when you begin debugging by pressing F5 or clicking the *One-Click Rails Debugger* icon.

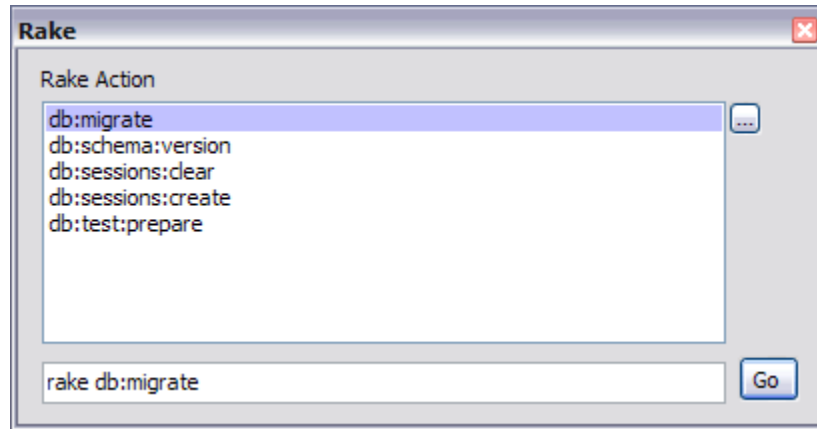
For step-by-step guidance on installing LightTPD for use with Cylon, refer to the documentation in the */Documentation* directory beneath your Ruby In Steel installation.

You can start your selected server by clicking the *Start Server* button on the Ruby menu or toolbar. Options to configure the server are available by selecting the *Tools* menu, then *Options, Projects and Settings*, Ruby In Steel. For more information, see [Configure Debug Script and Server Script](#).

## Rake

### [Developer Edition Only]

*Rake* has its own item on the Ruby menu (it also has its own 'leaf' icon on the Ruby toolbar). You may use this to display the Rake window which provides an alternative to running **rake** from the command prompt.



A default list of Rake actions is provided and you may select one of these by clicking it. The complete rake script is shown in the text field at the bottom of the window. This script can be edited. To run the script, press the 'Go' button.

**Note:** Rake files are run from a default directory. If you wish to run rake in a different directory, you can change the Rake Working Directory for the current project in the Project Properties page.

### CUSTOMIZING THE SCRIPT ACTION LIST

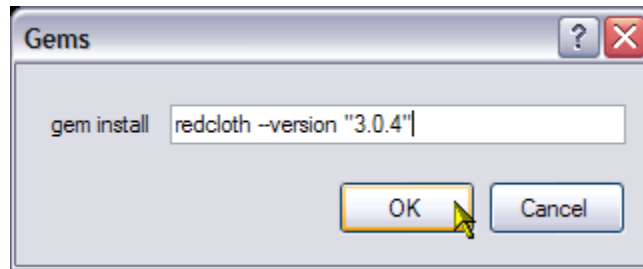
To change the available rake actions, click the [...] button at the top right of the Rake Action list and enter new actions, one per line, or delete existing actions. Click OK to confirm your edits. The edited Rake Action list will now be used as the default.

### PLACING THE RAKE WINDOW

If you need to use the Rake window regularly, you may wish to dock it in the Ruby In Steel environment. Right-click its caption bar, select 'Dockable' and drag it to place it in a dock site. If you use it irregularly or you wish to place it on a non-primary monitor, right-click and set to 'Floating'. Click the caption bar 'Close' button when you have finished using the window. You can also use the window as a tabbed document page.

## Gems

[Developer Edition Only]



Use the *Gems* dialog to attempt to find and install add-on Ruby Gems packages. Enter the name of the Gem and any additional parameters into the text field and click *OK*. So, for example, entering this...

```
redcloth --version "3.0.4"
```

...is equivalent to running the following command from the system prompt:

```
gem install redcloth --version "3.0.4"
```

## Synchronize

If the files and folders shown in the Solution Explorer are not 'up to date' with the files and folders in your project directory on disk, click the Synchronize button to update the Solution Explorer. The display may become unsynchronized if you add, move or rename files from 'outside' Visual Studio (for example, by using the Windows Explorer or running scripts from a command prompt). The Solution Explorer also has a Synchronize button.

By default, if the Solution Explorer contains more than one project and no project is selected, Synchronize operates on the active project only. If you wish to synchronize a different project, you can do so by selecting a specific project node in the Solution Explorer. You may select multiple projects by CTRL-clicking two or more project nodes. You may then synchronize the selected projects. In multi-language solutions (for example, if you have a mix of Ruby In Steel and C# projects) Synchronize ignores non Ruby In Steel projects.



## SETTING SYNCHRONIZATION OPTIONS

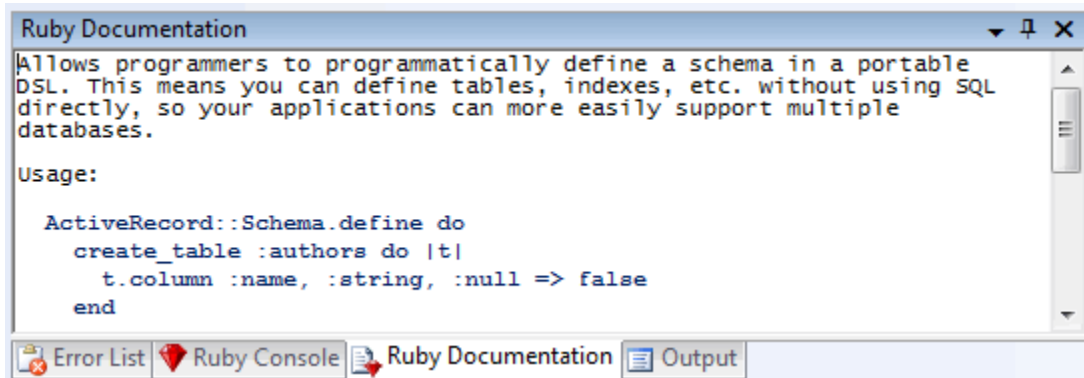
Before synchronizing, you may want to specify certain types of files and directories which you wish to exclude – for example, hidden directories or files with specific extensions. This will avoid cluttering the Solution Explorer by displaying files which you don't intend to edit or directories (such as backup or version-control directories) which form no part of the project.

To set these properties, select the *Project* menu then *Properties*. In the Properties dialog box, find the *Synchronization* group. To exclude files with no extension following a dot (such as Rails scripts, for example), set *Exclude Files With No Extension* to true. To exclude directories and files with the hidden attribute, set *Exclude Hidden Files and Folders* to True; to include them, set this property to False. To omit files with specific extensions, enter a semicolon-delimited list of file extensions into the *Exclude From Synchronization* field, like this:

**.log;.txt;.xxx**

**Note:** Certain file names cannot be added to the Solution Explorer (for example, file names containing ampersands '&' are not allowed). If you want to view all the files in the project directories on disk – including hidden files and directories, files with illegal names and files which have been excluded from synchronization, select *Project, Show All Files*. The *Show All Files* menu item is a toggle which displays or hides files which are not a part of the project. *Synchronization*, on the other hand, permanently adds files to the project (though these can be selectively removed by right-clicking in the Solution Explorer and selecting *Exclude From Project*).

## Ruby Documentation



This displays an RDOC window in which you can browse formatted documentation from any embedded RDOC comments above methods and classes. Hover over an identifier in the code editor in order to display any associated documentation.

## Reset Fonts and Colors

Selecting this option resets all the colors and fonts to their default values. This may be useful if you have changed the settings or if the fonts are not currently displaying correctly. You may want to save your current color scheme before using this option.

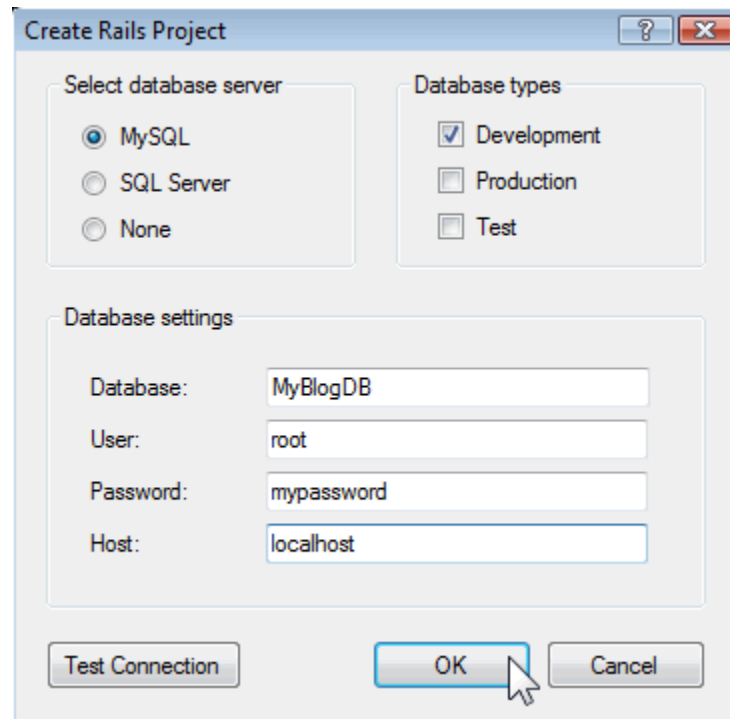
## Rails Development

RUBY IN STEEL CAN HELP YOU TO CREATE, IMPORT EDIT AND DEBUG RUBY IN RAILS APPLICATIONS.

### The Rails New Project Wizard

#### STEP 1 – START A NEW RAILS PROJECT

- > Select the *File* menu, *New, Project, Ruby In Steel, Rails Project*
- > Enter project name (e.g. **MyBlog**) in the *Name* Field
- > Optionally browse to set the location of the project
- > Optionally, select *Create Directory For Solution*
- > Click *OK*



#### STEP 2 – SET UP THE DATABASE

In the *Create Rails Project* dialog: In order to create a new database for your Rails application, enter a *name* for the database, a *user name* which should previously have been set up in your database server, a database *password* (if you have one) and a *host*. If you do not wish to create a database (if one already exists or you plan to create the database at a later stage), select *None* in the *Select Database Server* options. Otherwise, select *MySQL* or *SQL Server* (this latter choice should also be selected for Microsoft's SQL Express) and select one or more *Database types*.

**Note:** the database types correspond to the three types traditionally used by Rails. You may select one or more of these. Ruby In Steel automatically generates databases with the suffixes *\_development*, *\_production* and *\_test* and these suffixes are appended to the database name. For example, if you have named the database: *MyRailsApp* and selected Development and Test database types, Ruby In Steel will create two databases named *MyRailsApp\_development* and *MyRailsApp\_test*. All the necessary configuration information will also be written into a *database.yml* file as required by Rails. If you do not select a database server (if you chose *None*), a *database.yml* file will be created but you will need to edit it by hand in order to add details corresponding to any databases which you create.

## EXAMPLES OF CREATING DATABASES

### > MySQL

**Note:** You must have MySQL installed; it can be downloaded from <http://dev.mysql.com/>. This example assumes that you have set up a user name (here 'root') and a host (here 'localhost'). In this example, it is assumed you have left the database password unspecified. The database will be named 'MyBlog' and you may select the *Development*, *Production* and *Test* database types. This is what you would enter into the Database Settings fields of the *Create Rails Project* dialog:

Database: **MyBlog**  
User: **root**  
Password:  
Host: **localhost**

### > SQL Server

**Note:** You must have SQL Server installed; a free edition called SQL Express can be downloaded from <http://msdn.microsoft.com/vstudio/express/sql/>. This example assumes that you have set up a user name (here 'john'), a password ('secret') and a host (here '.\SQLEXPRESS'). The database is named 'MyBlog' and the *Development* database type only is selected. This is what you would enter into the Database Settings fields of the *Create Rails Project* dialog:

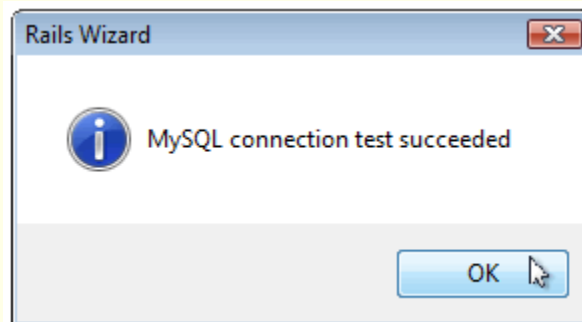
Database: **MyBlog**  
User: **john**  
Password: **secret**  
Host: **.\SQLEXPRESS**

### Verify The Database Connection

You may verify the connection to your database by clicking the '*Test Connection*' button prior to proceeding.

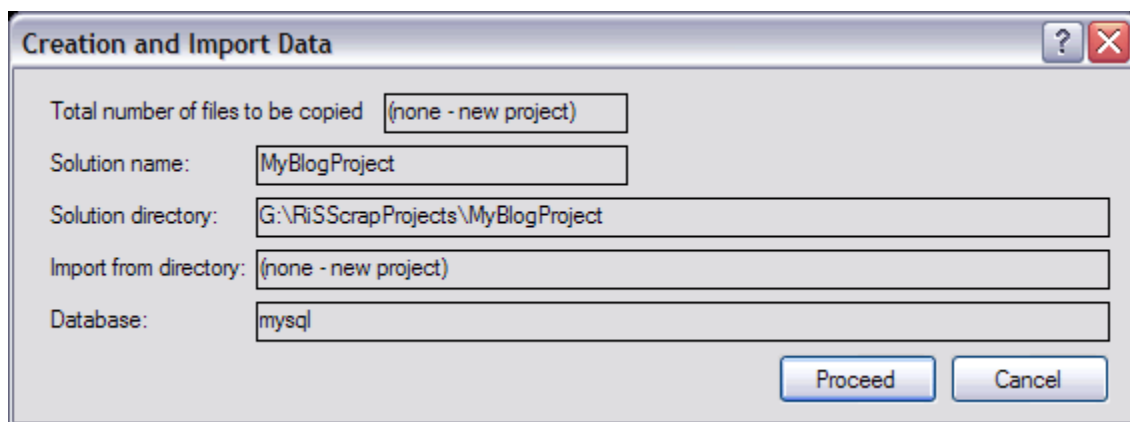


If a Connection is made, a message similar to the following will appear...



If the Connection cannot be made an error message will be shown. If this happens, you need to check that the paths to the database server are correct (*Tools, Options, Projects and Solutions, Ruby In Steel*) and that your user name, password and host details are all entered correctly in the Create Rails Project dialog.

After you click OK to close the *Create Rails Project* dialog, a page of information about the new project appears (this shows the Solution Name, the Solution Directory and Database).



You should verify that the details are correct and, if so, click *Proceed* to continue.

All being well, Ruby In Steel will now display all the files and directories of your new Rails application in the Solution Explorer. If you encounter any problems you may want to check that you have installed your database server correctly and that the user name, password and host which you entered into the Rails Project dialog match the details which you previously set up in the database server.

**Note:** Database creation will fail if you attempt to create a database with a name which already exists. Check the Visual Studio Output pane (press *CTRL+W, O*) to see any error messages.

## The ERb Editor and the Visual Rails Workbench

When working with Rails templates (files with the default extension *.html.erb* in Rails 2 and *.rhtml* in Rails 1) you have the option of using either the ERb Editor or the Visual Rails Workbench.

The **ERb Editor** provides basic editing features such as code coloring and collapsing and snippets. You may use this editor if you only need to make simple editing changes to Rails template files.

The **Visual Rails Workbench** is an integrated visual development environment which provides all the code editing features of the ERb Editor plus drag and drop page design tools, code navigation utilities and import/export features. You should use the Visual Rails Workbench if you need to make pixel-perfect changes to the design of your views or if you wish to export and import pages to external editors such as Adobe Dreamweaver or Microsoft Expression Web.

### TO SELECT YOUR ERB ENVIRONMENT.

#### 1) For all new projects:

Go to: *Tools, Options, Projects and Solutions, Ruby In Steel*.

Find the *Visual Rails Workbench* group.

Set the option for *Use Visual Rails Workbench* to *true* (or *false* to use ERb editor).

#### 2) For the current project only:

Go to: *Project, Project Properties*.

Find the *Visual Rails Workbench* group.

Set the option for *Use Visual Rails Workbench* to *true* (or *false* to use ERb editor).

You will need to close and reopen any View templates for the changed option to be enabled.

## The ERb Editor

The ERb Editor displays syntax coloring for HTML and embedded Ruby code. When working with complex HTML, the Visual Studio HTML editor is a better choice. The HTML editor supports embedded styles (CSS) and JavaScript coloring; HTML tag-matching using clickable tags beneath the editor and IntelliSense features such as completion lists for HTML attributes. Switch between the editors to take advantage of the special features of each. You will be prompted to save unsaved changes before switching.

- To switch to from the ERb editor to the HTML editor, press **ALT+H**
- To switch from the HTML editor to the ERb editor, press **ALT+J**

These shortcuts only apply to the ERb editor and not the Visual Rails Workbench.

## ERb Code Coloring And Folding

Rails *.erb* or *.rhtml* templates are displayed with syntax sensitive code coloring which supports the coloring both of HTML and embedded Ruby code. Code collapsing in *.erb* or *.rhtml* files is performed on HTML tags.

## Ruby On Rails IntelliSense

### [Developer Edition Only]

Providing IntelliSense for Ruby files in a Rails application poses a special challenge due to the fact that the relationships between Ruby On Rails code files are not specified by 'requiring' other files in the code itself. Related code files are only 'wired together' when the Rails system processes them after the application is deployed. This explains why the Ruby interpreter itself is unable to run Ruby On Rails code files in the normal way.

Ruby In Steel addresses this problem by attempting to work out the implicit relationships between Ruby On Rails code files at design time so that the IntelliSense system can then gain access to the appropriate methods to display in code completion lists. For example, if your application includes a controller that descends from **ApplicationController** you may press *CTRL+Space* to view **ApplicationController** and **ActionController** methods. It is not always possible to infer the actual types of instance variables in Rails code files.

**NOTE:** You may add additional IntelliSense to Rails (including Database-specific IntelliSense) using the [IntelliSense Librarian](#).

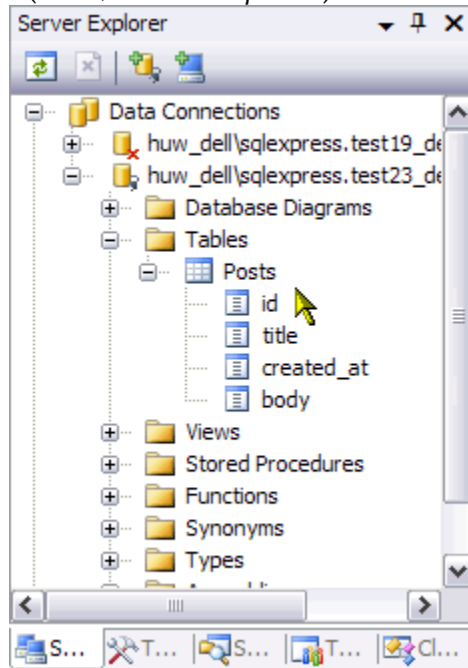
If you need access to files which are not automatically required, you can also specify additional files to be required and parsed for IntelliSense. Do this by selecting the file for which you need IntelliSense in the Solution Explorer; then add the full paths to the required files in the *Require Files* property of the Properties panel. Each file path must be separated by a semicolon. A quick way of obtaining the path to a file is to open that file in Visual Studio, click its editor tab and select *Copy Full Path*. This path can then be pasted into the *Require Files* property field.

**Note:** In order for Ruby On Rails files to have access to IntelliSense, you must ensure that the **Rails Project** property is set to **true** (in the *Project Properties* dialog). This property is set to true by default when you create, import or convert a Rails project. If you change the value of the *Rails Project* property you should close and reopen any files open in the editor in order to reinitialize their IntelliSense.

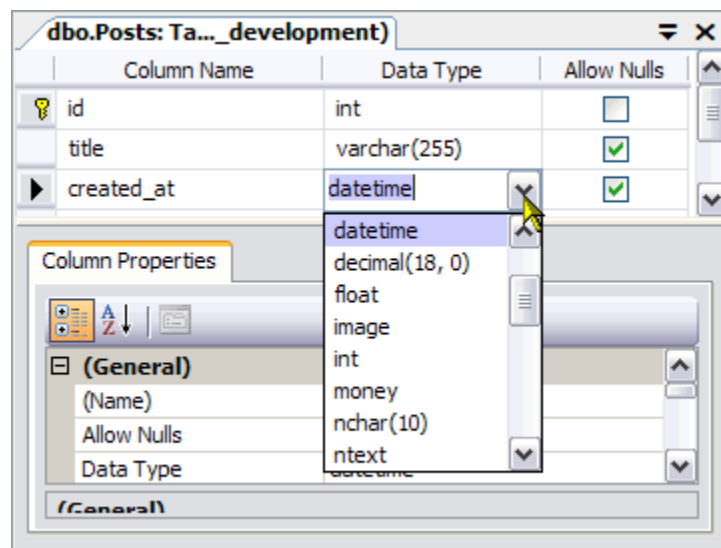


## Integrated SQL Server Development

If you use Microsoft's SQL Server (or the free SQL Express) as your database server you can create and edit tables right from within the Visual Studio environment itself. Open the database in the Server Explorer (*View, Server Explorer*)...



..and edit the tables and column properties...



## The Visual Rails Workbench

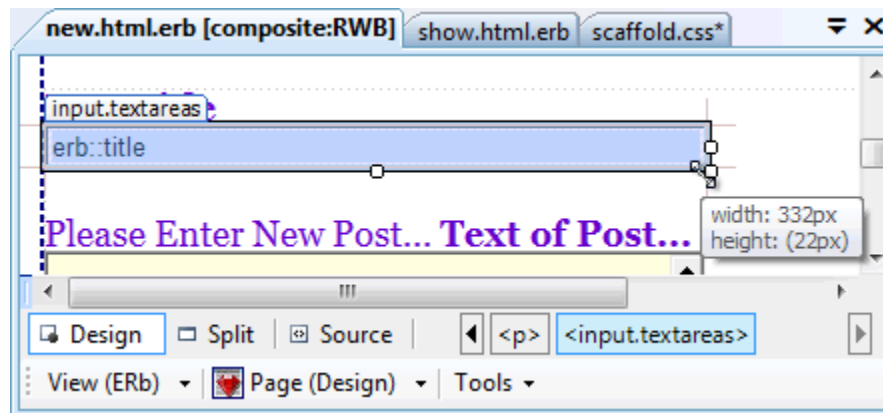
*[Developer Edition Only]*

The Visual Rails Workbench is an integrated coding and visual design environment for Rails applications. Your Rails pages can be edited in two ways...

### Rails Templates and Full Page Design



**Rails Templates:** By default, page designs in Rails are defined in the form of ‘document fragments’ (Layouts, Views and Partials) comprising a mix of Ruby and HTML code. These are not valid HTML pages and cannot be edited using a normal web page designer. When a Rails application is run, Rails executes the embedded Ruby and assembles an HTML page by processing and combining multiple template files.



**Full Page Design:** The Visual Rails Workbench processes and combines the ERb templates at design-time - in effect emulating the runtime behavior of Rails in order to provide an editable ‘composite’ HTML page derived from the ERb templates. This gives you the ability to move and resize existing controls and adjust properties for pixel-perfect design using Visual Studio or a third-party web editor such as Adobe Dreamweaver.

**Note:** The Visual Rails Workbench only operates on the *design and layout* elements of a page. It does not interact with any ‘model’ data referenced by embedded Ruby code.

## Features of the Visual Rails Workbench

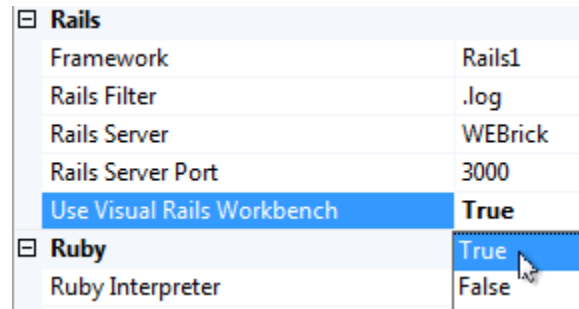
- Full page editing of complete web pages (“composites” of Rails layouts/views/partials)
- Drag and Drop design - add controls from a toolbox
- Set properties using the Property panel
- Resize and move controls using mouse or keyboard
- Split view code/form editing
- Toggle ERb/RHTML editing between HTML editor and Rails (Ruby-aware) editor
- Round-tripping between ‘web format’ HTML and ‘Rails format’ ERb/RHTML
- Edit code as ERb/RHTML or as HTML
- Document Navigator navigates document structure (HTML)/or methods (ERB/Ruby)
- Quick navigation between controller and view
- Import/Export to other web page design tools
- Save/restore named ‘versions’ of page designs to/from an archive of work in progress
- Auto-backup of changes to templates
- Support for Rails 1 and Rails 2

## Enable The Visual Rails Workbench

The Visual Rails Workbench may be enabled selectively for the currently active project or globally for all new projects. When it is not enabled, the much simpler (non-visual) ERb code editor will be used for editing Rails templates.

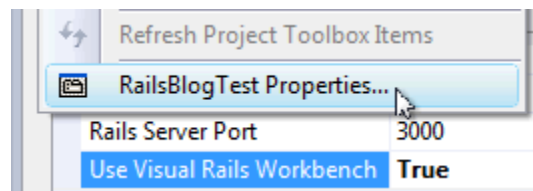
### ENABLE FOR ALL NEW PROJECTS

To make the Visual Rails Workbench the default for your Rails projects, select *Tools, Options, Projects and Solutions, Ruby In Steel* and ensure that the option, *Use Visual Rails Workbench* is set to *True*. Every new project will now use the Workbench. This option will not affect the currently loaded project.



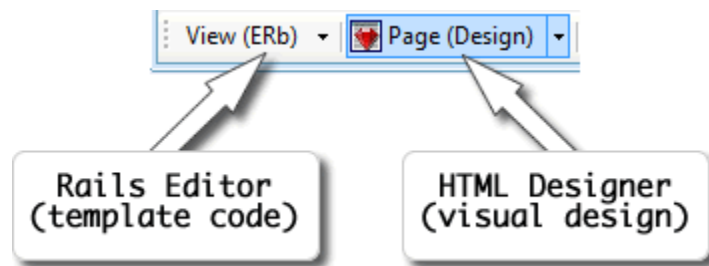
### ENABLE FOR CURRENT PROJECT

To enable the Visual Rails Workbench for the currently loaded project, select *Project, Project Properties* and ensure that the option, *Use Visual Rails Workbench* is set to *True*. If you are currently editing any View templates, close these and reopen them to edit them in the Visual Rails Workbench.



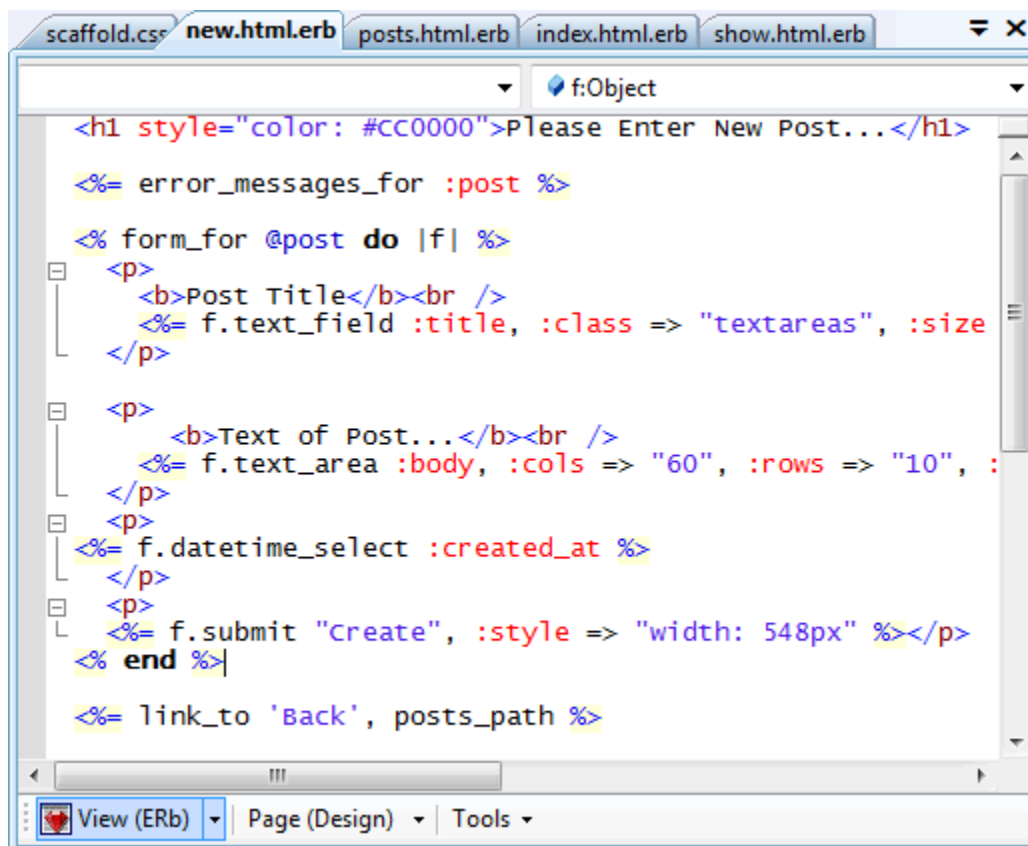
## The Visual Rails Workbench Environment

There are two linked coding and design workspaces: the *Rails Editor* and the *HTML Page Designer*. You may toggle between these workspaces by clicking one of the buttons at the bottom-left of the Visual Rails Workbench window:



### THE RAILS EDITOR

You may edit 'native Rails template' code (*.rhtml* or *.html.erb* views, partials and layouts) in the Rails Editor...



## THE HTML PAGE DESIGNER

When you need to work with the visual design, you can switch to the HTML Page Designer...



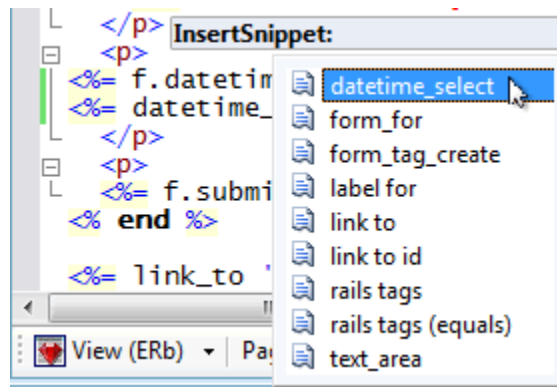
- The currently active workspace (Rails Editor or HTML Page Designer) is indicated by an icon on one of the two buttons at the bottom of the workspace.
- The text on the Rails Editor button indicates the type of template file being edited - (Layout, View or Partial).
- The HTML Page Design button is only displayed when a View template is loaded - a View is required in order to construct a full HTML page for the visual designer.

## The Rails Editor

The Rails Editor is where you will enter and edit the code of Rails templates. If you don't need to use the visual design features of the Rails Workbench, you will probably spend all your time working here.

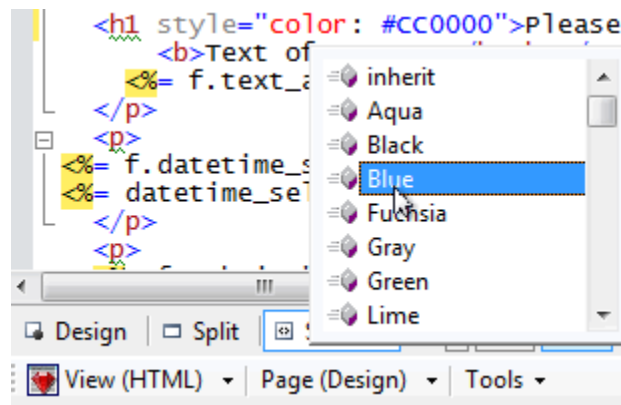
The Rails Editor has two editing modes - one of which is optimized for embedded Ruby, while the other is optimized for editing HTML.

### ERB Mode



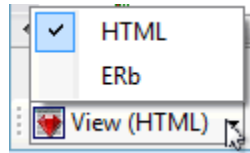
This gives you access to the features of the Ruby In Steel embedded Ruby editor including color coding of Ruby code, snippets and quick navigation (right-click) between a controller and a view.

### HTML Mode



This gives you access to the features of the Visual Studio HTML editor such as code completion for JavaScript and styles.

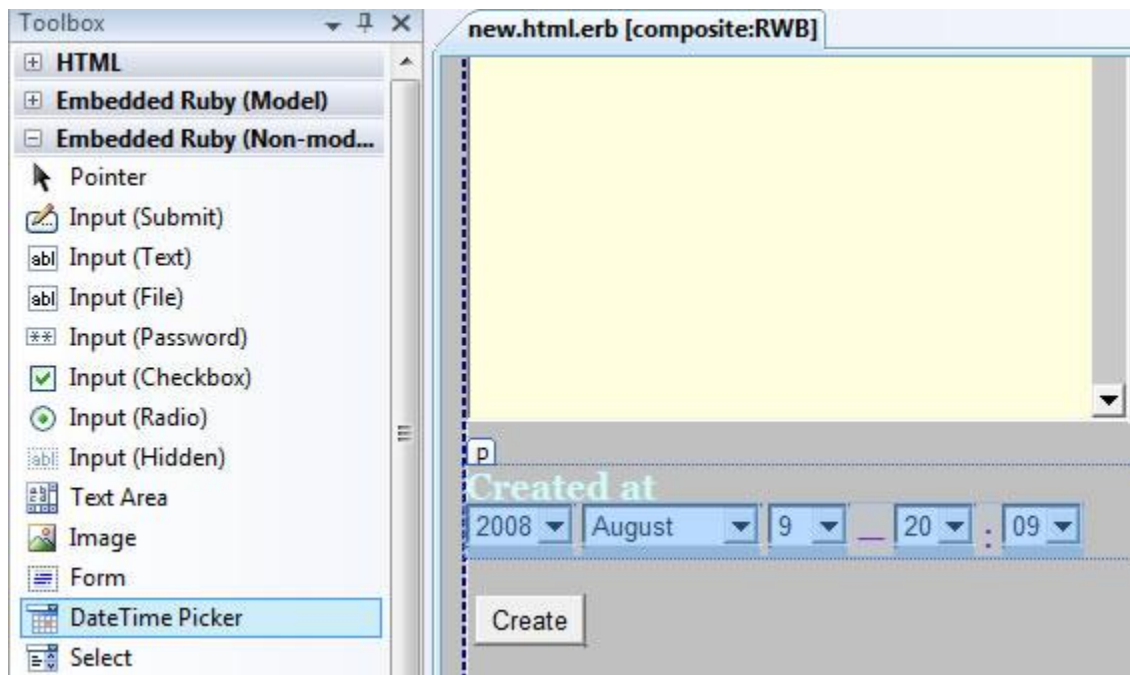
## TOGGLE RAILS EDITOR MODE



To toggle editing modes, click the arrow-head selector to the right of the Rails Editor button.

## The HTML Page Designer

The HTML Page Designer allows you to edit the loaded view in HTML format. This displays HTML controls - such as buttons and Text Areas and allows you to add new controls by dragging and dropping from the Toolbox.



Three groups of tools are available:

- **HTML** - Standard HTML controls
- **Embedded Ruby (Model)** - Rails data-aware controls that will be translated to ERb
- **Embedded Ruby (Non-model)** - Rails controls that will be translated to ERb

See [Rendering Visual Components](#) for more details on these controls.



## Constructing An HTML Page From Rails (ERb) Templates

When you switch to the HTML Page Designer, the View is automatically inserted into the Layout template to which it belongs. Any Partial's required by the View are also inserted in the appropriate locations. The Rails Workbench then 'translates' the embedded Ruby code and (where possible) substitutes the HTML which corresponds to that code. For example, if an ERb view template contains this...

```
<%= f.datetime_select :created_at %>
```

...when loaded into the HTML Page Designer, the following code is substituted:

```
<span erb:name="erb:datetime_select" erb:blockvar="erb:f"
erb:method="erb::created_at" ><select id="f_:created_at_1i" name="f[:created_at(1i)]">
<option value="2002">2002</option>
<option value="2003">2003</option>
<option value="2004">2004</option>
```

...(etcetera)

Note that the Rails Workbench defines the **erb** namespace to keep track of translated elements such as `<span erb:name="erb:datetime_select" ...>`

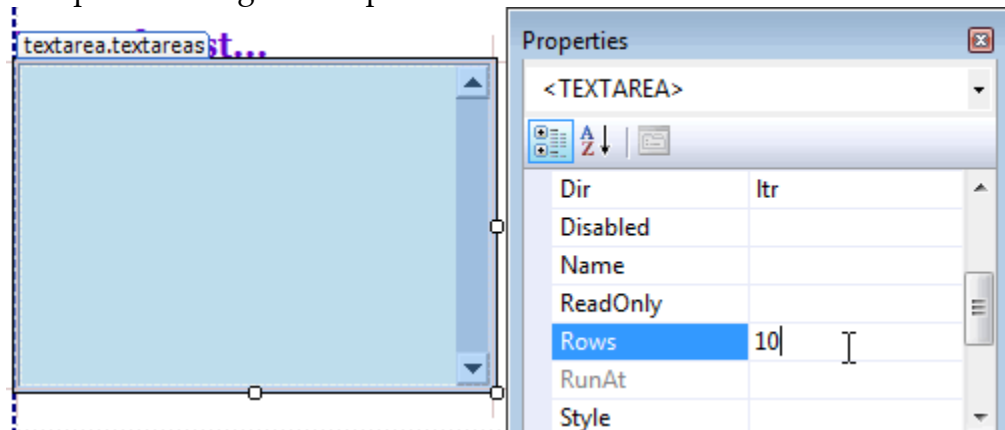
**IMPORTANT:** You must not edit any elements containing references to the **erb** namespace!

When you move from the HTML Page Designer back to the Rails Editor, the HTML code is translated back to ERb format (HTML templates containing embedded Ruby).

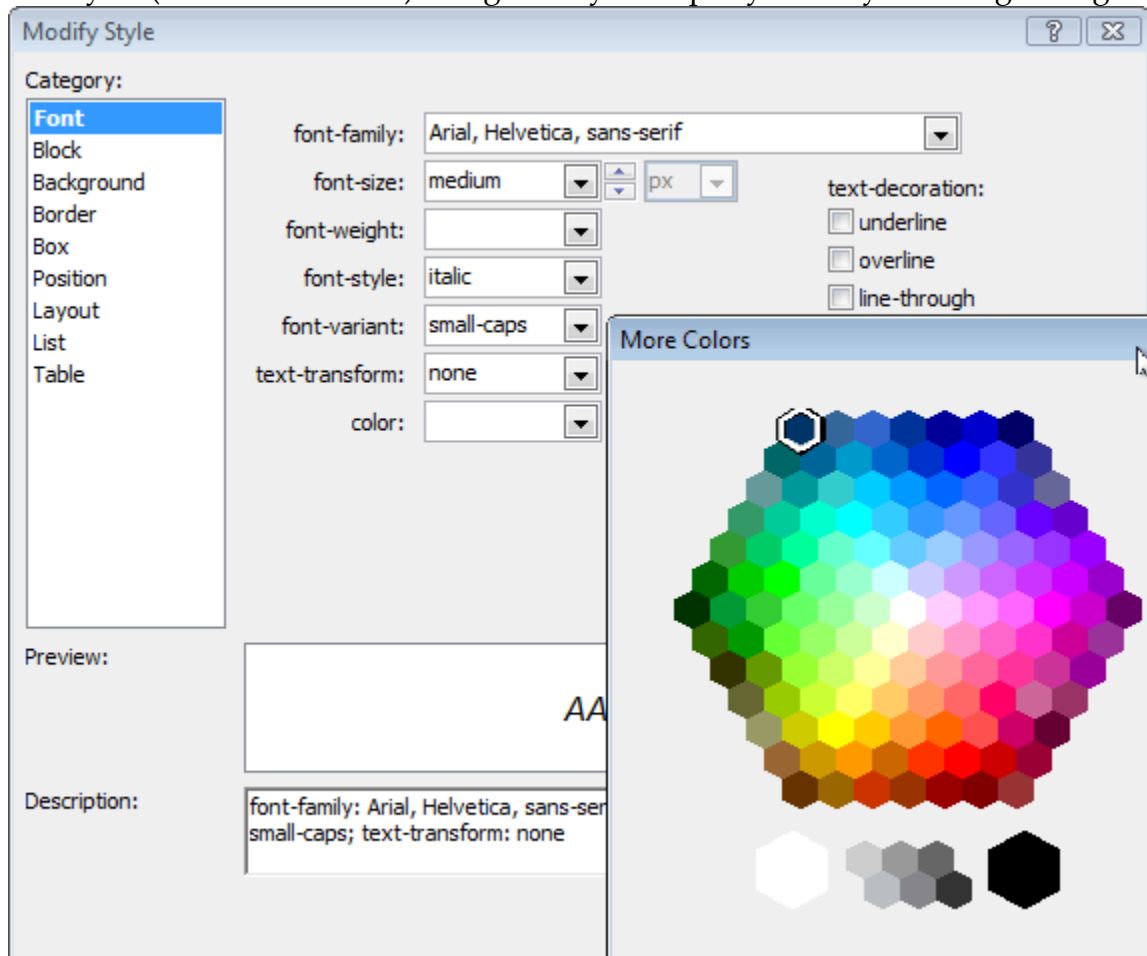
## Using The Visual Page Designer

In the HTML Page Designer, you may:

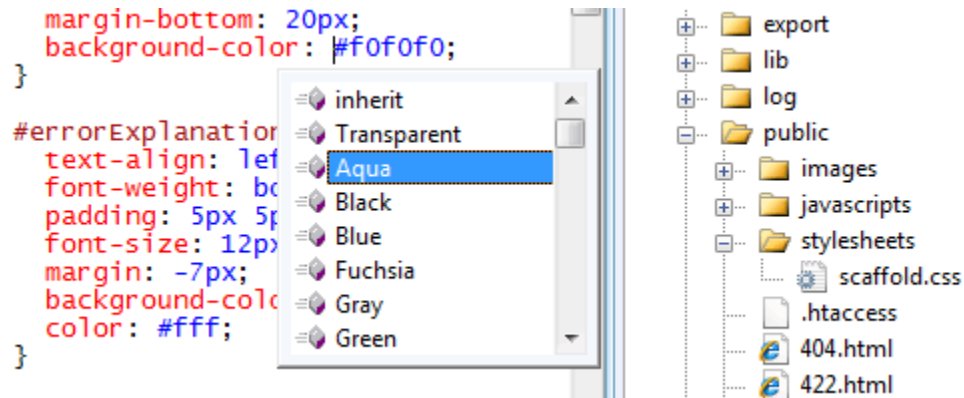
- Resize controls by dragging using the mouse.
- Alter Properties using the Properties Panel...



- Set Styles (in the current file) using the Style Property and Style editing dialogs...



- Set Styles in an attached CSS Style Sheet by loading CSS file into editor...



- Toggle Design/Split/Source

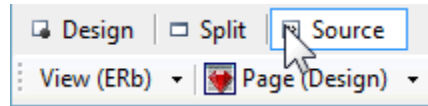


The HTML (Page Designer) has three buttons: **Design**, **Split** and **Source** - to select the visual designer, a split view (Designer and HTML code editor) or the HTML Code Editor alone.

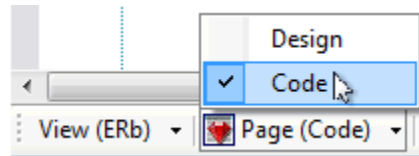
You may also click tag elements to the right of these three buttons (e.g. `<form>` `<p>` etc.) to navigate through the tags surrounding the currently selected element.

## Page Design (Code) View

There are two ways of viewing the HTML code of a page design.



In the **Page (Design)** view you may click *Source* or *Split* to edit the code using Microsoft's HTML Editor. this gives you access to editing features such as HTML and CSS Style code-completion.



Switch to **Page (Code)** view the code arranged in color-coded blocks highlighting those elements of the HTML page which were generated from specific Rails (ERb) templates: for example, a Layout, a View and one or more Partials.

```
49 </div>
50 </div>
51
52 <h1>My Blog</h1>
53 <p>Enter a post</p>
54 <a erb:name="erb:link_to" erb:code="erb:83">
55 <a erb:name="erb:link_to" erb:code="erb:83">
56 </div>
57
58 </body>
59 </html>
```

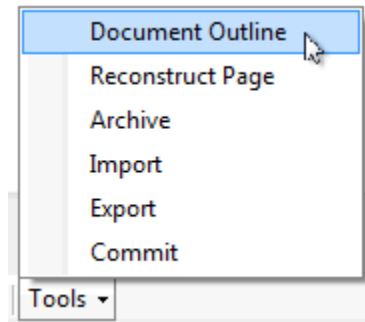
If you wish to edit code in **Page (Code)** view (shown above), double-click one of the colored areas to place that area into edit mode. You will be prompted to commit any changes when you move back into the Rails **View (ERb)** editor.

## The Visual Rails Workbench Tools

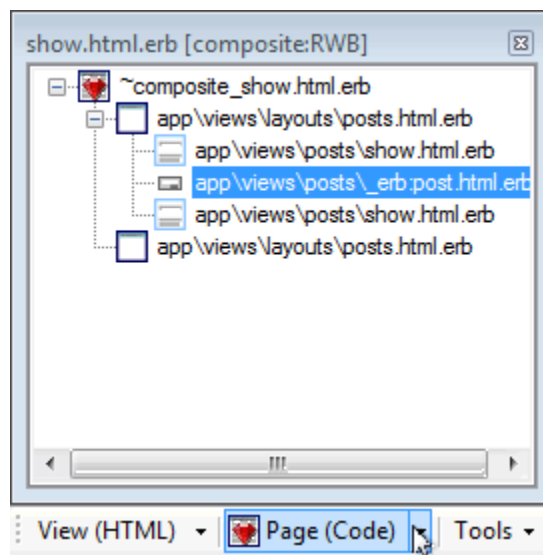
The Visual Rails Workbench *Tools* menu provides access to a number of tools. The items on the *Tools* menu vary according to the active editing view. The largest number of items are available in the **HTML Page Designer** (both the *Design* and the *Code* view)...

## The Document Outline

The Document Outline is a tree-structured window that shows you a hyperlinked outline of the currently active document. To view the Document Outline, select *Tools, Document Outline* (alternatively, select the *View* menu then *Other Windows, Document Outline*):

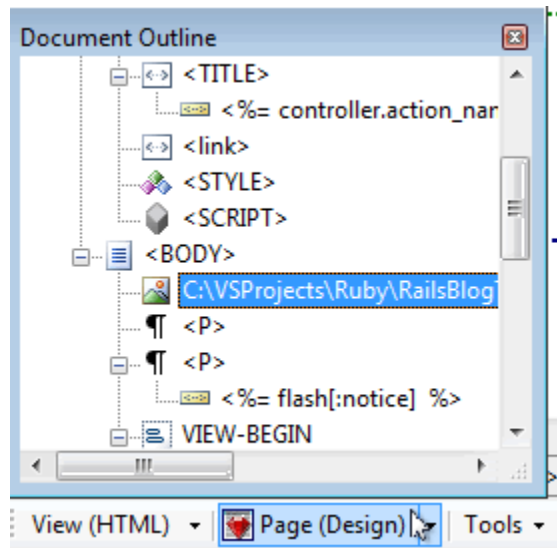


## HTML PAGE DESIGNER - CODE VIEW



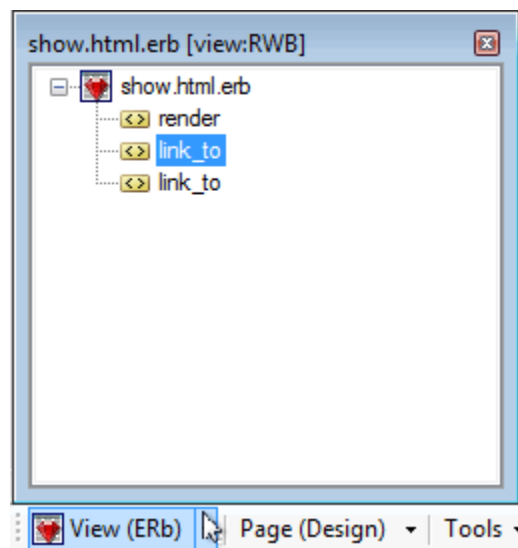
In *Page Designer (code view)*, the Document Outline shows component parts of the composite HTML web page corresponding to Rails templates (Layout, View, Partials) defining this page.

## HTML PAGE DESIGNER - DESIGN VIEW



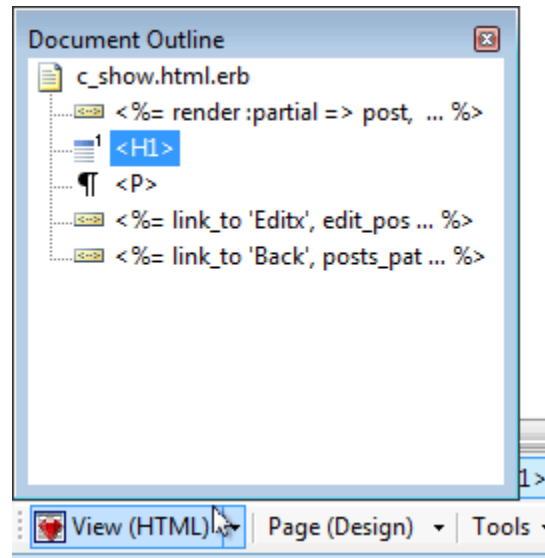
In *Page Designer (design view)*, the Document Outline shows the HTML structure of the web page.

## RAILS EDITOR (ERB MODE)



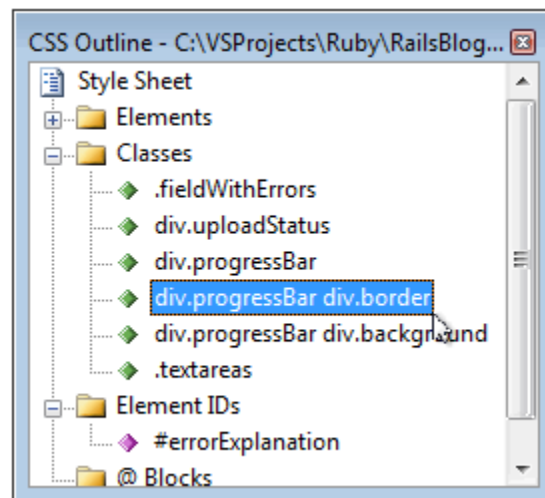
When you view Rails Templates as ERb in the Rails Editor, the Document Outline shows all the embedded Ruby (code between `<%..%>` and `<%=..%>` tags). It does not show other HTML tags.

## RAILS EDITOR (HTML MODE)



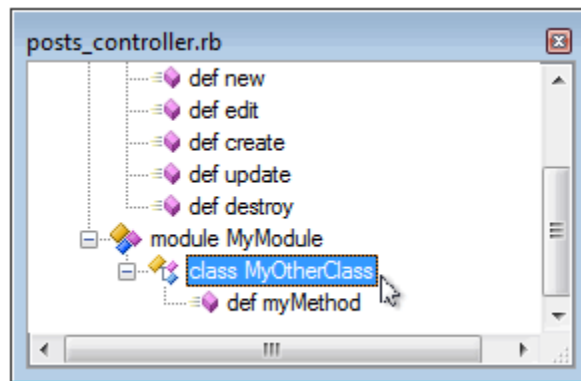
When you view Rails Templates as HTML in the Rails Editor, the Document Outline shows all standard HTML tags (including embedded Ruby).

## CSS STYLESHEET



When a stylesheet (with the extension `.css`) is loaded, the Document Outline shows styles grouped by category - such as Elements, Classes and Element IDs.

## RUBY



When working in a Ruby (.rb) code file, the Document Outline shows the names of modules, classes and methods.

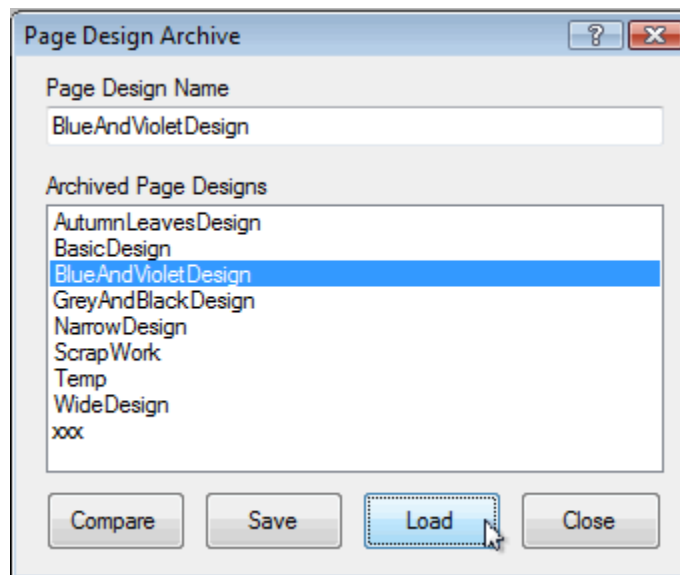


## Reconstruct Page

If you wish to scroll back any changes you have made but which have not yet been committed, select *Reconstruct Page*. This will rebuild the page from the ERb template files (Layout, View, Partials) which define it.

## Archive

Use the Archive to save ‘working backups’ of your page designs. You can save named designs into the Archive so that you may save and reload named versions of your designs. Archived designs are saved under the `\~intermediate` directory in your Rails project. The `\~intermediate` directory is reserved for Ruby In Steel ‘working’ files and is not displayed in the Solution Explorer. You should regard archives files as temporary. To save final page designs, you should use Export.



Before reloading an archived design, you may click the *Compare* button to view the differences between the current page and the archived version. You will need to have a ‘merge’ or ‘differencing’ tool installed in order to compare files. There are several such tools freely available such as Restore previous versions of backed up Rails view templates or whole HTML format Rails web pages using the Visual Rails Workbench and a differencing tool such as WinMerge (<http://winmerge.org/>) and DiffMerge (<http://www.sourceforge.com/diffmerge/>). A Differencing Tool may be installed globally in the [Ruby In Steel Options](#) page or for just the loaded project in [Project Properties](#).

## Archives Don't Restore? Check Your, Styles!

If it seems that changes which you've made to page are not restored when you load a design from an archive, you may want to check if some styling information is being saved into a separate stylesheet file rather than into the HTML of the page itself. For example, let's assume that you have assigned a class to a text area. If you look at its tag in the HTML code, the text area will begin like this:

```
<textarea erb:name="erb:text_area"...
```

Somewhere between the opening < and the closing > delimiter may be a style ('class') name:

```
class="textareas"
```

This class may be defined in a linked stylesheet (a file with the extension .css), like this...

```
.textareas {  
  background-color: #FFFFFFDF;  
  width: 721px;  
}
```

Now when you change properties of the text area - such as its width, these changes will be saved into the stylesheet, not into the HTML of the page itself. When archived pages are loaded, the styles in the existing stylesheet will be applied to the text area so that there will be no visible change due to reloading the archive. If you wish the changes to be saved into the HTML, remove the styles (the classes) from the HTML tags or place the styles into the HTML page itself rather than into a separate stylesheet. Alternatively, you may wish to save multiple named versions of your stylesheets to allow you to load and test different sets of styles.

## Import

Use this option to load a previously saved 'composite' HTML page based on the current view. For example, if you are working on a composite of **edit.html.erb** you wish to reload a design saved with the name **classic\_edit\_page.html**. When you reload the page, it will completely replace the page design in the editor. We recommend that you always save your current design prior to importing a saved design so that you can easily restore it if you wish.

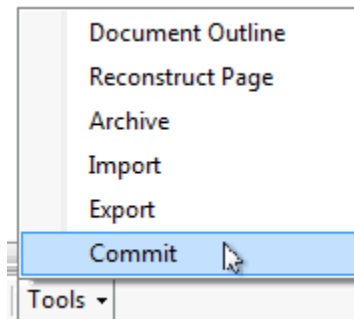
## Export

Use this option to save the current 'composite' HTML page. We recommend using a descriptive name when saving to allow you easily to identify a design if you decide to reload it later on.

By default, page designs are saved under the `\export\Page Design` directory in the current Rails project. This directory retains the relative links to other directories containing images and styles. This is convenient if you need to work on a design using a third-party web design tool. You may export to any other location but if you do so, you may lose the links to styles and images.

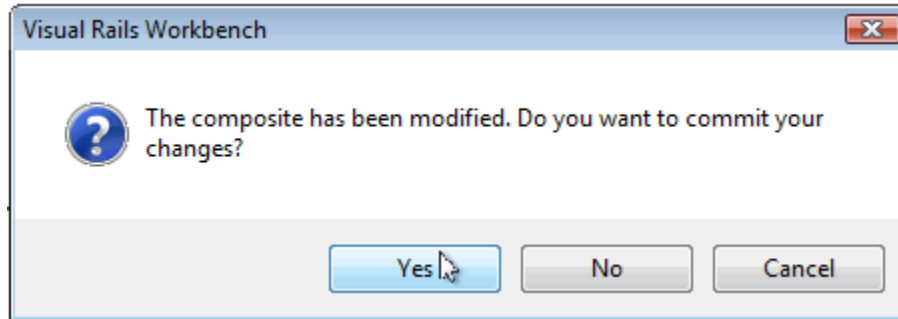
## Commit To Save Changes

When you have made changes to a page design in the HTML Page Designer you need to commit to save those changes back to the original Rails ERb templates from which the page was generated.



To Commit Changes, select *Tools, Commit*.

If you leave the HTML designer (by switching back to the Rails Editor) and there are unsaved changes, you will be asked whether or not you wish to commit those changes.



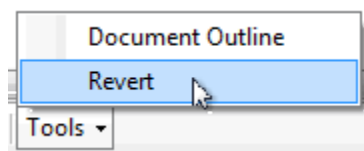
**IMPORTANT:** You must *commit* your changes in order to disassemble and translate an HTML page into its component Rails-format template files. Saving the page using *File, Save* will only save a temporary copy!

## Rails Editor Tools

When you are working in the Rails (View, Layout or Partial) Editor, the *Tools* menu lists two options: *Document Outline* and *Backup*. The Document Outline has been described previously.

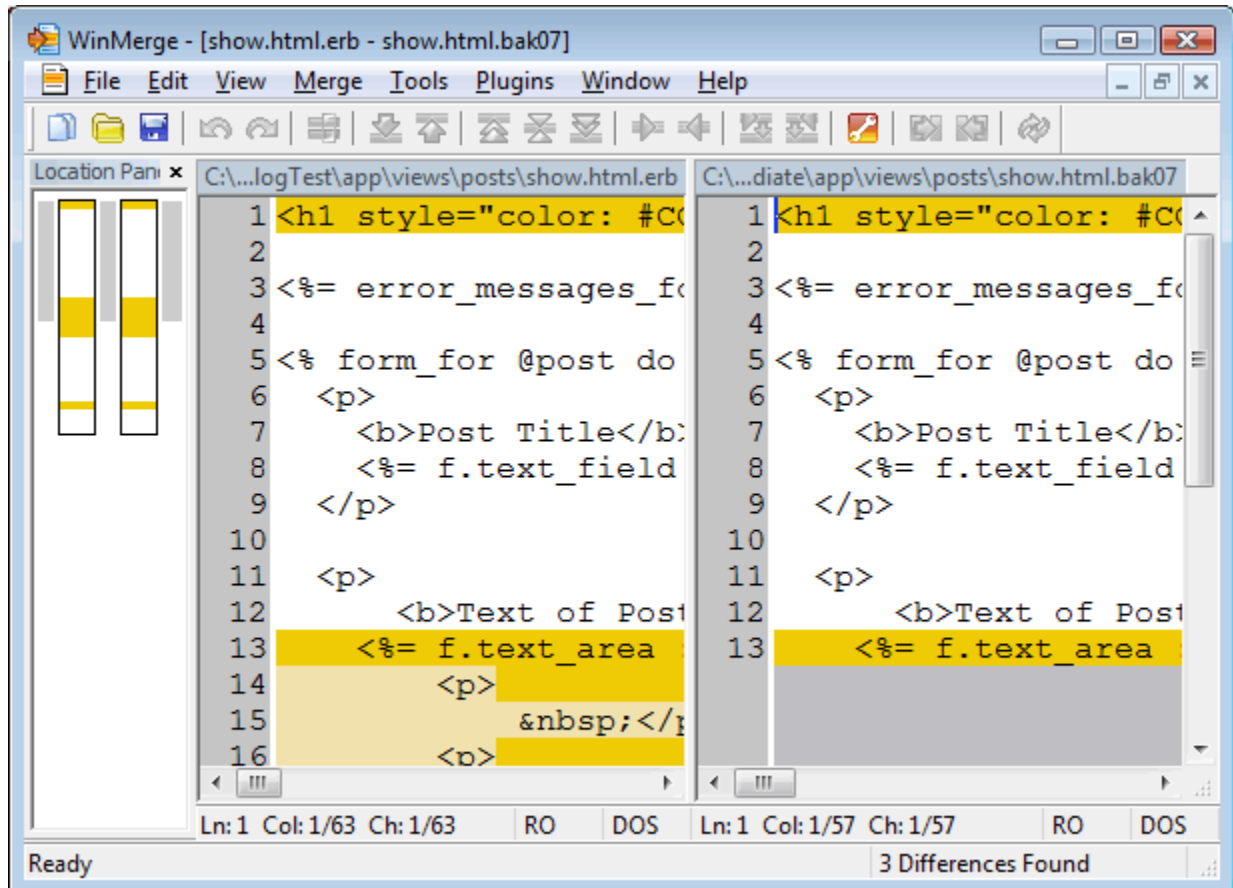
## Rails Editor Backups

Automatic incremental backups of your ERb-format templates are saved when changes are made (these may result either from edits made to the template code or from changes committed in the HTML Page Designer). If you wish to backtrack to an earlier version of a specific Rails-format template file (for example, a view template called **show.html.erb**), you should select *Tools, Revert* in the Rails Editor, pick a backup from the list and click the Revert button.



You may select and preview a specific version of the template before reverting...

As with [Page Design Archives](#), you may compare the differences between the currently loaded template and a backup prior to restoring the backup. You need to have a differencing tool installed in order to do this...



Here a backup is being compared using the WinMerge differencing tool.

### Archives or Backups?

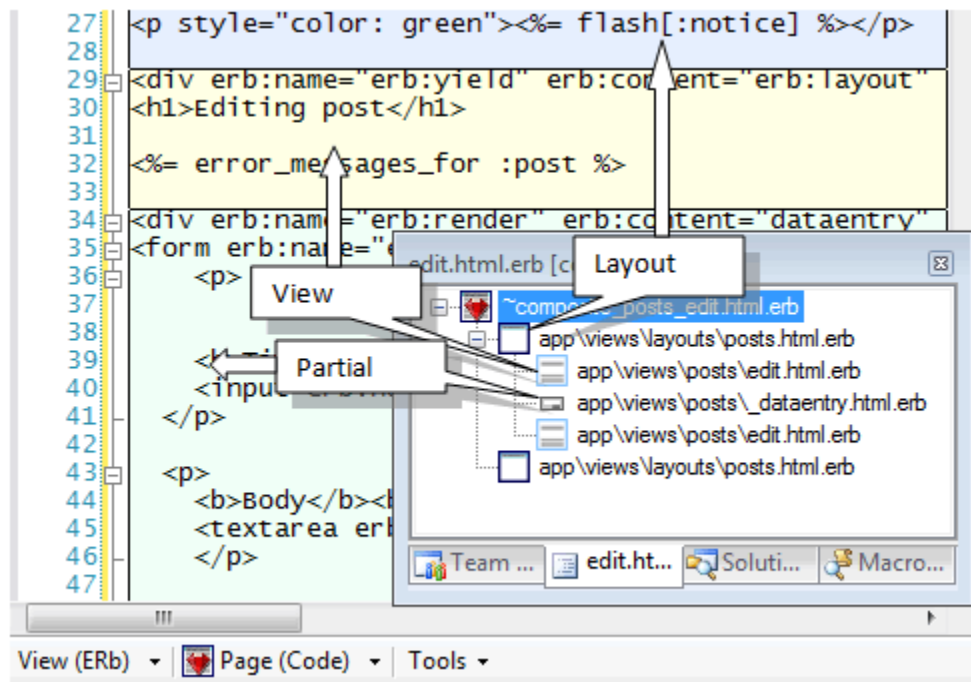
The Rails Workbench automatically backs up changes made to Rails (ERb) templates. Changes to the HTML page designs are not automatically backed up but can be explicitly saved into the Archive. The Archive and the backups are complementary: if you **restore a backup** of a Rails-format template (such as a View), *only that one selected template file will be restored*. A new HTML page will be constructed using this restored template when you switch to the HTML Page Designer. On the other hand, if you **load a named HTML page design from the Archive**, *all the Rails template files* (Layout, View, Partials) corresponding to that page design will be generated when you commit your changes.

## ERb / HTML Round-Tripping - How It Works

When you move from the Rails Editor to the HTML Page Designer, the original code of your Rails Templates is assembled and translated in order to form a complete, editable HTML 'composite' page.

To understand how this works, let's assume that you load the View template, **edit.html.erb** into the Rails editor. When rendered as a web page in the running application, this View will be 'contained' by a Layout template called **posts.html.erb** and it will itself contain a Partial called **\_dataentry.html.erb**.

The Visual Rails Workbench constructs a representation of this *runtime* web page *at design-time*. It does this by inserting the Partial into its containing View and inserting the View into the Layout. It then translates any embedded Ruby code which relates to design and layout into the corresponding HTML. The end result is that you now have an editable HTML page that was created from several ERb-format Rails templates. If you switch to the *Page(Code)* editor and view the Document Outline (from the Workbench *Tools* menu), the structure of this 'composite' HTML page becomes clear. The HTML areas which correspond to the original Layout, View and Partial templates are color-coded in the editor and shown on branches in the Document Outline:

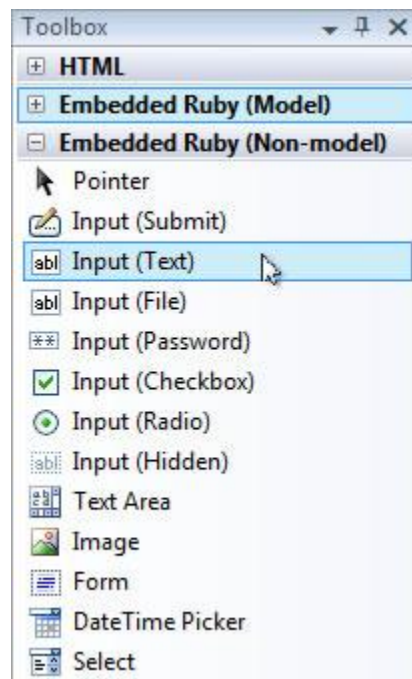


Here in the *Page (Code)* editor, you can navigate the 'composite' HTML page which has been constructed from Rails (ERb) Layout, View and Partial templates. The corresponding areas of the HTML page are color-highlighted in the editor and shown as branches in the Document Outline.

## Rendering Visual Components

HTML ‘components’ - interactive elements such as text edit areas and buttons - may either be defined in the original Rails templates or they may be added in HTML Page Design Editor by dragging and dropping components from the Toolbox onto the web page.

The Visual Rails Workbench takes care of translating these components to and from HTML and ERb format when you switch between working in the HTML Page Design Editor and the Rails code editor.



There are three types of component in the Toolbox, arranged in groups (**HTML**, **Embedded Ruby (Model)** and **Embedded Ruby (Non-model)**):

## HTML COMPONENTS

These are normal HTML components and no translation is done when moving between the HTML Page Designer and the Rails Editor.

### *Example*

In HTML Designer:

```
<input id="Text1" type="text" />
```

In Rails Editor:

```
<input id="Text1" type="text" />
```

## EMBEDDED RUBY (MODEL) COMPONENTS

These are ‘Rails-aware’ controls. When you move from the HTML Page Designer to the Rails Editor, they will be translated into embedded Ruby format including dummy ‘block form variables’ (by default, `f`).

### *Example*

In HTML Designer:

```
<input erb:blockvar="erb:f" erb:method="erb::myName"  
  erb:name="erb:text_field" name="erb:f[:myName]" size="30" type="text"  
  value="erb::myName" />
```

In Rails Editor:

```
<%= f.text_field :myName, :size => "30" %>
```



## **EMBEDDED RUBY (NON-MODEL) COMPONENTS**

These are ‘Rails-aware’ controls. When you move from the HTML Designer to the Rails Editor, they will be translated into embedded Ruby format but will not include ‘block form variables’.

### *Example*

In HTML Designer:

```
<input erb:name="erb:text_field_tag" name="TextFieldName" type="text" />
```

In Rails Editor:

```
<%= text_field_tag "TextFieldName" %>
```

Note that when you have made changes to a form design in the HTML Page Designer you will be prompted to save or ‘commit’ those changes when you switch to the Rails Editor.

## Supported Commands and Limitations

The Visual Rails Workbench translates the following Rails commands to and from HTML:

- check\_box
- check\_box\_tag
- content\_for
- datetime\_select
- file\_field
- file\_field\_tag
- form\_for
- form\_tag
- hidden\_field
- hidden\_field\_tag
- image\_submit\_tag
- image\_tag
- javascript\_include\_tag
- link\_to
- password\_field
- password\_field\_tag
- radio\_button
- radio\_button\_tag
- render
- select
- select\_tag
- stylesheet\_link\_tag
- submit
- submit\_tag
- text\_area
- text\_area\_tag
- text\_field
- text\_field\_tag

The range of supported commands will be extended in future revisions and we welcome suggestions and requests from users.

The Visual Rails Workbench does not currently support sequential commands. For example, this ERb code...

```
<%= link_to image_tag %>
```

...will translate `image_tag` as text in the HTML editor. However, this will be correctly retranslated into the original ERb code when you return from the HTML editor to the Rails editor.

### ‘Ignore Markers’ (`erb:ignore`)

If you wish specific Rails commands to be passed between the Rails and HTML editors verbatim (without being translated), you should include the text `"erb:ignore"` inside the tags. This is called an ‘ignore marker’. You may remove ignore markers prior to deploying your application. If, on the other hand, you wish to leave an ‘ignore marker’ in place you should ensure that it is syntactically correct. Only when it forms part of correct syntax will Rails be able to process the command at runtime. For example, let’s assume that you want the following to be passed to the HTML editor verbatim...

```
<%= link_to posts_path %>
```

The syntax of the Rails `link_to` command requires the link name as its first argument, and optional hash of options and HTML options as the second and third arguments. The third argument would be the best place in which to put the ignore marker, like this:

```
<%= link_to posts_path, {}, { :ignore => "erb:ignore" } %>
```

This will be passed verbatim to the HTML editor. When the Rails application is deployed, the resulting HTML will be rendered like this:

```
<a href="/posts/2" ignore="erb:ignore">/posts</a>
```

As far as the Visual Rails Workbench is concerned, the `erb:ignore` marker may be placed at any point between the enclosing Rails `<% tags %>`. However, unless it forms part of correct Rails syntax, the command may not be processed accurately at runtime. That is why it is best to adhere to valid Rails syntax.

Any items associated with the **erb:ignore** marker (such as the symbol **:ignore** in the example above) are optional and are only required for compatibility with Rails syntax. If you wish the runtime HTML generation to validate, you may wish to use standard HTML attributes - for example, use **:id** instead of **:ignore**:

```
<%= link_to posts_path, {}, { :id => "erb:ignore" } %>
```

## Debugging Rails Applications

Ruby In Steel provides debugging for Rails applications in addition to 'pure Ruby' programs. The Developer Edition features two debuggers – the fast 'Cylon' debugger and the slower Ruby debugger.

Dedicated support is provided for the following web servers:

- > WEBrick
- > Mongrel
- > LightTPD

### Which Server Should You Choose?

It is entirely up to you which server you use. WEBrick or Mongrel may already be installed with your Rails installation and, in most cases, it will be easier to use one of these two servers. LightTPD, on the other hand, may be somewhat faster and it is favored by some users. LightTPD does require additional effort to install and configure, however.

### How To Start The Rails Debugger

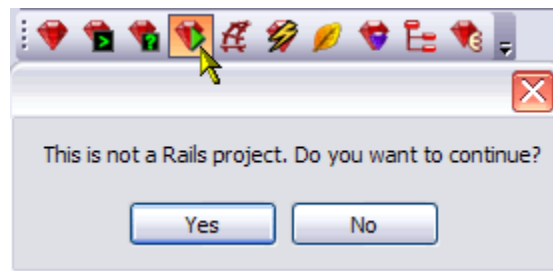
There are two ways in which a Rails debugging session can be started:

- Click The One-Click Rails Debugger (in the Rails menu or toolbar) [*Developer Edition Only*]
- Press F5 (or select Start Debugging from the Debug menu)

## WHICH TO USE - F5 OR THE ONE-CLICK RAILS DEBUGGER?

In most cases, it makes no difference whether you start Rails debugging using F5 or the *One-Click Rails Debugger*.

However, bear in mind that the *One-Click Rails Debugger* ***always assumes that you wish to debug a Rails application***. Consequently, if the *Rails Project* property is set to *False*, it will ask you whether you wish to debug the project as a Rails application anyway.



However, when the *Rails Project* property is False, F5 adopts the default behavior for a Ruby (non-Rails) project and attempts to run the currently active Ruby source file.

If you are debugging a Rails application but need the option to debug selected Ruby files too, set the *Rails Project* property to False. Use F5 to debug a selected Ruby file; use the *One-Click Rails Debugger* to debug the current Rails project. Otherwise (if you only need to debug the Rails project) make sure that the *Rails Project* property is set to True (this is the default for Rails projects) and start debugging using either F5 or the *One-Click Rails Debugger* as you prefer.

For information on setting the *Rails Project* property, refer to: [Project Properties](#)

## Create A Rails Project...

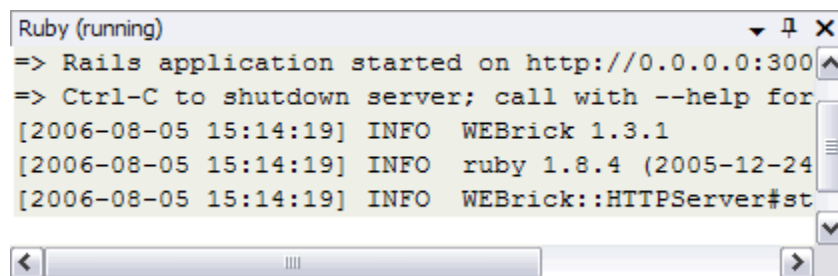
Create a Ruby On Rails project in the usual way (select *New|Project|Ruby In Steel* and click *Rails Project*).

## Select A Server...

You may select your server in *Tools|Options|Projects and Solutions|Ruby In Steel* (for all future projects) or *Project|Properties* (for just the current project). In the Rails group find *Rails Server* and select the desired option. Optionally you may also change the *Rails server port* (this affects Mongrel and WEBrick only). Its default value is 3000. If this port is already in use (say by another server), set it to some other values (such as 3003).

## Click and Debug...

Set breakpoints in any Ruby source files. Then, to start debugging, press F5 or select *One-click Rails Debugger* from the *Ruby* menu. After a few seconds (be patient, this is not instantaneous!) the server will start up and display output similar to the following in the Ruby console...



```
Ruby (running)
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for
[2006-08-05 15:14:19] INFO WEBrick 1.3.1
[2006-08-05 15:14:19] INFO ruby 1.8.4 (2005-12-24)
[2006-08-05 15:14:19] INFO WEBrick::HTTPServer#start
```

### **Problems...?**

- If you can't see the Ruby Console, you can load it by selecting the *View* menu, *Other Windows*, then *Ruby Console*.
- If error messages appear in the Ruby Console, check that you have correctly set all the properties as explained earlier and that another instance of the WEBrick or Mongrel server is not already running. Fix the problems and try again...
- If a message pops up saying "*This is not Rails project. Do you want to continue?*", this mean that a property indicates that this is not a Rails application. Click "Yes" to run it as Rails anyway. Or [set a project property](#) to mark this as a Rails project.

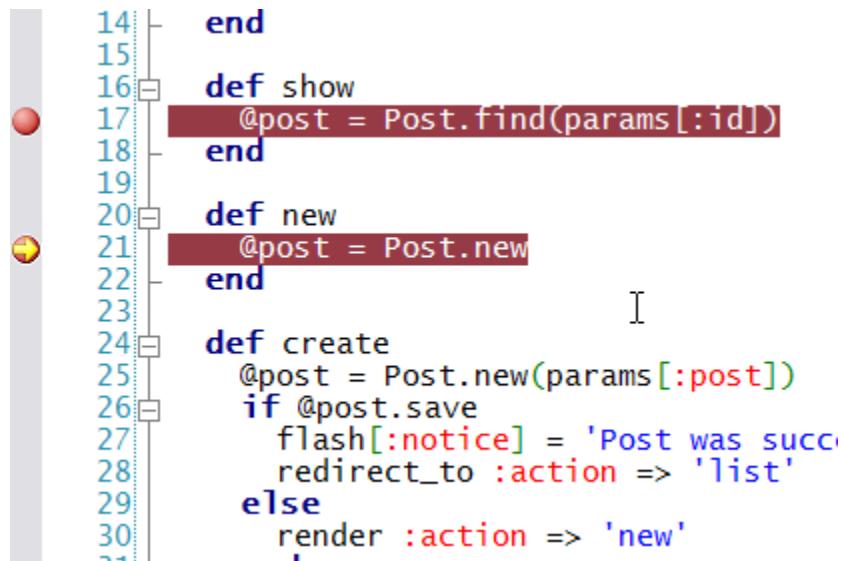


## Run Rails Application In Web Browser

Once you have started debugging a Rails application, you may load up a Web Browser in order to interact with it (this may either be a standalone browser or the integrated browser in Visual Studio – you can load this from the menus: *View, Other Windows, Web Browser*).

In the address bar, enter the full address to your Rails application just as you would if running it normally. For example, you might enter: **http://localhost:3003/blog** – that is, the full address, including the host name and port, to your application. Let's assume you put a breakpoint on a method which is called when you enter a new post; you now click the *New post* link in the web browser which runs the bit of code you want to debug...

### STOP ON BREAKPOINT



The image shows a code editor window with a vertical line on the left side representing the line numbers. A red circle breakpoint is set on line 17, and a yellow circle breakpoint is set on line 21. The code is as follows:

```
14 | end
15 |
16 | def show
17 |   @post = Post.find(params[:id])
18 | end
19 |
20 | def new
21 |   @post = Post.new
22 | end
23 |
24 | def create
25 |   @post = Post.new(params[:post])
26 |   if @post.save
27 |     flash[:notice] = 'Post was succ
28 |     redirect_to :action => 'list'
29 |   else
30 |     render :action => 'new'
31 |   end
32 | end
```

The result is that the debugger stops on the line with the breakpoint...

## DEBUG

Watch 1			
Name	Value	Type	
@post	nil	Nil	
list	[...]	Array	
[0]	<0x5029428>	Post	
@attributes	{...}	Hash	
body	"Today I took the dog for a long walk"	String	
title	"My Wonderful New Post"	String	
id	"1"	String	
created_at	"2006-12-15 20:47:00"	String	
[1]	<0x50293ec>	Post	
[2]	<0x50293b0>	Post	
@headers	{...}	Hash	

And now you can use all the usual Ruby In Steel debugging features to trace through the code and monitor variables and expressions in the various debugging windows.

## Debug ERb

In addition to debugging 'pure Ruby' (for example, in controllers), you may also place breakpoints on embedded Ruby in Rails templates. When you stop at a breakpoint, all the usual debugging features are available.

```

10 <% for post in @posts %>
11   <tr>
12     <td><%=h po
13     <td><%=h po
14     <td><%=h po
15     <td><%= lin

```

post <db:row>

- id: 4
- title: "xxx"
- body: "yyy"
- created\_at: "2008-01-30 11:25:00"
- updated\_at: "2008-01-30 11:25:35"

%></td>  
:delete

Watch 1			
Name	Value	Type	
@posts	[...]	Array	
[0]	<db:row>	Post	
id:	2	Fixnum	
title:	"Hurrah! My 2nd post!"	String	
body:	"I'm really picking up speed no	String	
created_at:	"2008-01-27 10:49:00"	String	
updated_at:	"2008-01-27 10:49:58"	String	
[1]	<db:row>	Post	
[2]	<db:row>	Post	

Watch 1 Generate Rake Output Autos

## Customization

### Code Color Options

Ruby In Steel provides syntax sensitive coloring for Ruby (.rb) and Rails (.erb and .rhtml) files. In Ruby source files. The following global coloring options assigned in Visual Studio are used when appropriate: **Strings**, **Comments**, **Numbers**, **Keywords**, **Identifiers** and **Plain Text**, plus a range of other 'system-wide' options such as *Selected Text*, *Inactive Selected Text*, *indicator margin*, *bookmark*, *breakpoint* and so on. There are additional Ruby-specific coloring options. Rails template files (.erb and .rhtml) have their own coloring options for HTML tags (the global Visual Studio HTML colors are not used) while embedded Ruby in .erb and .rhtml files adopts your selected Ruby-specific colors. All colors can be set in the *Options* dialog (*Environment, Fonts and Colors*) available from the *Tools* menu.

<b>Ruby Brace</b>	colors of ( ), { } and [ ]
<b>Ruby Class Definition</b>	colors of class name declarations (e.g. class MyClass)
<b>Ruby Class Variable</b>	colors of class variable
<b>Ruby Constant</b>	colors of Constant names
<b>Ruby Global Variable</b>	colors of Global variables
<b>Ruby Instance Variable</b>	colors of Instance variables
<b>Ruby Method Call</b>	colors of Method name reference (e.g. aMeth)
<b>Ruby Method Definition</b>	colors of Method name declarations (e.g. def aMeth)
<b>Ruby Module Definition</b>	colors of Module name declarations
<b>Ruby Regular Expression</b>	colors Regular Expressions
<b>Ruby Symbol</b>	colors of symbols such as :mysymbol
<b>Rails Attribute Name</b>	colors of HTML attributes such as width or bgcolor
<b>Rails Attribute Value</b>	colors values of HTML attributes
<b>Rails Element Name</b>	colors of HTML elements such as <td> or <head>
<b>Rails Entity</b>	Colors elements such as &nbsp;
<b>Rails Layout</b>	Layout color in Rails Workbench (Code View)
<b>Rails Partial</b>	Partial color in Rails Workbench (Code View)
<b>Rails Serverside Script</b>	colors of <% <%= and %> tags
<b>Rails Tag Delimiter</b>	colors of Rails tag delimiters
<b>Rails View</b>	View color in Rails Workbench (Code View)

## Color Options For Other Windows

Many other Visual Studio windows may be selected from the *Show Settings For* drop-down list in the *Options* dialog, *Environment/Fonts and Colors* pane. For example, here you can alter the colors of the Watch and Locals window or *All Text Tool Windows* such as the Output window. To alter the colors of the interactive console, select *Ruby Console* from the list.

## Options (Projects and Solutions)

You may set a number of global options which will become the defaults for all new Ruby In Steel Projects in the *Options* dialog (*Tools, Options, Projects and Solutions, Ruby In Steel*). Note that changes made to these options will only affect *new* projects. Existing projects will retain the options that were in force at the time of their creation. These global options are a subset of the options available as Project Properties. The Project Properties always override the global options.

Please refer to the [Project Properties](#) documentation for more information.

Global Options include the following items which are not available as Project Properties:

### DATABASE

- > MySQL Path  
The optional path to MySQL (e.g. `\mysql\bin`)
- > SQL Server Path  
The optional path to SQL Server (e.g. `\Tools\bin`)

(**Note:** You may optionally use other database servers with your Rails applications but these will not be automatically supported by the [Rails New Project Wizard](#)).

### RAILS

- > Rails Filter  
A semicolon delimited list of file extensions which you wish to exclude from the Solution Explorer when creating a Rails project - e.g.: `.log;.txt`

## Editing Options

The Ruby and Rails editing options are set under the *Text Editor* branch of the *Options* dialog (available from the Tools menu). Use the Ruby options to set defaults for Ruby code and the ERb options (optionally) to override some of these settings when working with Rails (*.erb* or *.rhtml*) templates.

### GENERAL OPTIONS

These include a variety of options which are listed for all Visual Studio languages. Where certain options are not applicable, they are grayed out. For Ruby In Steel, the relevant options are:

- > **Auto list members** – to provide IntelliSense code completion
- > **Parameter information** – to provide IntelliSense Parameter tips
- > **Enable Virtual Space** – enter code (or comments) into any blank area of the editor
- > **Word Wrap** – causes long lines to wrap automatically
- > **Show Visual Glyphs for Word Wrap** – displays a small graphic where text wraps
- > **Apply Cut or Copy** – cut or copy blank lines (when disabled, blank lines are ignored)
- > **Line Numbers** – displays line number in the left-hand margin
- > **Enable single-click URL navigation** – URLs become hyperlinks (*CTRL+LeftMouseButton*)

### TABS

Here you can set code Indenting and Tabbing options (*Developer Edition only*):

- > **None** – no automatic indenting options will be applied
- > **Block** – when you press Enter, the new line will align with the line above it
- > **Smart** – when you press Enter, the new line will align according to syntax

You can also specify the Tab and Indent sizes in this dialog and set an option to determine whether Tabs are entered as Tab characters or as sequences of space characters.

### FORMATTING

- > Automatically format complete block on 'end'  
Format code between a Ruby keyword and end after end is entered
- > Automatically add 'end' after 'class', 'def' or 'module'  
Insert end keyword automatically after class, def or module

## Ruby IntelliSense Options

You can specify a variety of options to configure the IntelliSense features available in your Ruby code files (*Developer Edition only*).

### COMPLETION LISTS

#### [Developer Edition Only]

- **Show Completion List After Character Is Typed**

The completion list will appear when alphanumeric characters are typed and any matching text pattern will be selected in the list.

- **Show Keywords**

Keywords such as **def**, **class** and **module** will be included in the completion list.,

- **Show Globals**

Global variables (beginning with \$) will be included in the completion list.

- **Show Class Names & Constants**

Class names and constants will be included in the completion list.

- **Show Snippets**

Code snippets will be included in the completion list.

- **Enter key is a commit character**

Inserts selected item into code editor on *Enter* (*Tab* is the default).

### SNIPPETS

- **Expand Snippets By Tab Character**

When this is selected, you can enter the shortcut of a snippet and press the Tab key in order to cause the snippet to be inserted into the code editor.

## **INTELLISENSE MEMBER SELECTION**

### **[Developer Edition Only]**

#### ➤ **Display Object Methods**

When this is selected, the methods defined in Ruby's Object class will be included in completion lists along with the methods of descendant classes. You may wish to disable the display of Object methods in order to limit the size of completion lists.

#### ➤ **Display Ancestor Methods**

When this is selected, the methods defined in Ancestors of the current object's class will be included in completion lists.

## **RDOC DISPLAY**

### **[Developer Edition Only]**

#### ➤ **Display RDOC in Tooltips**

When selected any documentation entered into a comment block above a class, module or method definition will be displayed in a tooltip when the mouse pointer hovers over an identifier of the appropriate type in the editor.

## **PARSING**

### **[Developer Edition Only]**

#### ➤ **Require File Depth**

This value determines the number of 'levels' of required files which are parsed in order to provide IntelliSense. If the value is 0, no required files are parsed. If it is 1, only files required by the current code file are parsed; if it is 2, files which are required by the files required by the current file are also parsed and so on.

When setting the 'require file depth', bear in mind that there is a trade-off between completeness and efficiency. Some Ruby class libraries may require files to many levels. In such a case the greater the required file depth, the more complete the IntelliSense provided. However, the IntelliSense Inference Engine consequently has to do far more work. As a result, the speed at which the code completion lists are updated to reflect any editing changes may deteriorate. As a general rule, we suggest accepting the default Require file depth of 2.

➤ **Provide IntelliSense While Debugging**

When this option is enabled, background parsing continues while debugging to ensure that IntelliSense and code formatting information is kept up to date. So (for example), if you enter new code while debugging, code-folding will be applied to it; if you debug into other source files, code completion information will be available. If this option is disabled, only the code formatting and completion information which was already parsed *prior* to debugging will be available. You may wish to disable this option to ensure the fastest response time during debugging



## Automating Ruby In Steel With Macros

**Ruby In Steel Developer** comes with a simple Macro library and a keyboard settings file which you can load up in order to assign some 'hotkeys' to selected macros.

The Macro Library contains a number of Visual Basic functions to help you to customize Ruby In Steel by adding various additional editing and project management features for Ruby and Rails projects. These include macros to add and remove user-defined pairs of tags to and from marked blocks, to paste Ruby code and enclose the pasted code <% and %> tags in Rails RHTML files, and macros to move quickly to and from between a Rails Controller and its associated View.

A library of utility macro functions is supplied to assist in customizing the existing macros.

The Ruby In Steel macro library is located in the */Extras/Macros* folder beneath your Ruby In Steel installation. A separate guide, **MacrosandKeyboardSettings.pdf** (in the */Documentation* folder beneath your Ruby In Steel installation) provides a detailed guide to using and customizing the macro library.

## JRuby and IronRuby

In addition to supporting the standard Ruby (1.8.6) interpreter, Ruby In Steel Developer also supports the Sun Microsystems' Java-based JRuby interpreter and Microsoft's IronRuby for the Dynamic Language Runtime on .NET.

### JRuby Support

When JRuby is selected as the *Project Type*, you will be able to run your programs in the docked interactive console and debug them using JCylon - a JRuby version of the Cylon debugger. Most of the debugging features of Cylon are also provided by JCylon but we do not guarantee absolute parity between the two debuggers.

#### TO SELECT JRUBY

For the current project:

- Select *Project, Properties*. Find the Ruby group and set *Ruby Type* to *JRuby*.

For all new projects:

- Select *Tools, Options, Projects and Solutions, Ruby In Steel*. Find the Ruby group and set *Ruby Type* to *JRuby*.

#### SETTING UP JRUBY

JRuby is not included as part of the standard installation of Ruby In Steel. In order to use JRuby you need to have Java and the Java SDK (version 6) installed as well as JRuby itself.

SapphireSteel Software does not offer support relating to the installation of JRuby. It is the responsibility of the user to ensure that a working installation is in place *before* attempting to use JRuby from within Ruby In Steel.

## INSTALL THE JDK

Before you trying to run JRuby, you must have Java and the Java Development Kit (JDK) installed and operational. If you haven't got Java and the JDK you should go to the Sun Microsystems Java site to download and install everything you need: <http://java.sun.com/> Be sure to install JDK 6. Earlier versions of the JDK are not supported.

## INSTALL JRUBY

For the latest downloads of JRuby follow links on the JRuby site at <http://jruby.codehaus.org>. Download one of the Zip archives containing the JRuby binaries (e.g. *jruby-bin-1.1.zip*) and extract it into a directory on your local hard disk (e.g. *C:\jruby-1.1*). I shall refer to this directory as the 'JRuby directory' - it is the directory which contains the first level of JRuby subdirectories - notably *\bin* and *\lib*.

**Note:** You may have problems if you install JRuby in a directory whose name includes a space character. To avoid this, be sure to avoid spaces in the directory name.

You now need to set an environment variable, called JRUBY\_HOME to point to the JRuby directory. You can do that from the *Windows Environment Variables* dialog. To load this, select *Start Menu, Settings, Control Panel*. In the Control Panel, double-click *System* to show the System Properties dialog. Click the Advanced tab or link, then the Environment Variables button. Beneath System Variables (or User variables if you plan to set this up for a named user), click the New button. Now enter JRUBY\_HOME as the Variable name and the full path to the installed JRuby directory (in my case, that's *C:\jruby-1.1\* – be sure to add the trailing '\') as the variable value. Click OK.

Now add the JRuby\bin directory to the system path. You can either add the full path or you can simply add bin to the end of the JRUBY\_HOME variable. You can do this once again from the 'Environment Variables' dialog. Select the PATH variable; click the 'Edit' button, scroll to the end of the existing path, add a semicolon followed by the path to the JRuby\bin directory. In my case, I added:

```
;%JRUBY_HOME%bin
```

Click OK to close the dialog. Then click OK to close the Environment Variables dialog.

You can now test it out by opening a command prompt (*Start Menu, Run*, Enter **cmd** and click OK) and entering...

**jruby -v**

It is possible, at this point that you may see a message stating:

*You must set JAVA\_HOME to point at your Java Development Kit installation*

If so, you need to verify that you have installed the Java Development Kit and that there is a variable specifying its directory. This variable is set up in the same way we set up JRUBY\_HOME earlier. In the Environment Variables dialog, click the New System Variables button and enter JAVA\_HOME as the variable name, followed by the path to the JDK. For me that happens to be *C:\Program Files\Java\jdk1.6.0\_05\*. Click OK and OK again to close the two dialogs.

Open a new command prompt (don't try to re-use any open command windows as they will not automatically adopt any changes made to the environment) and, once again, enter:

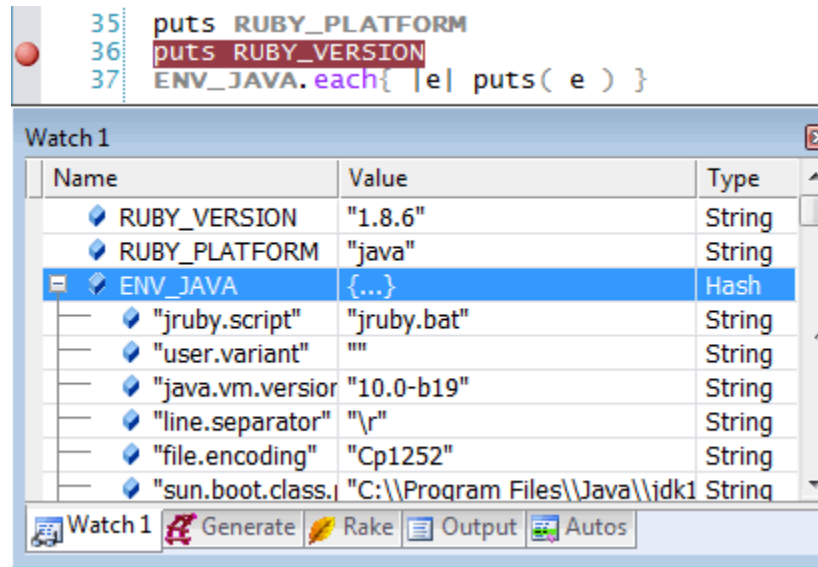
**jruby -v**

All being well, JRuby should bow respond with its version number, like this:

**ruby 1.8.6 (2008-03-28 rev 6360) [x86-jruby1.1]**

If not, then you need to verify all the preceding steps: Java and the JDK must be installed, JRuby with its ready-to-run libraries ('.jar' files in its *\lib* directory) must also be installed and the various environment variables and paths must be set up as explained above. Once you get JRuby to respond with its version number you are ready to use it in Ruby In Steel. Note, if Ruby In Steel was running at the time you set up your Java or JRuby environments, you will need to restart Visual Studio before using JRuby with your projects.

## VERIFY YOUR INSTALLATION



As explained earlier, you will need to set your Ruby Type to JRuby. To verify that JRuby is indeed being used, create a simple Ruby (.rb) file and evaluate the following:

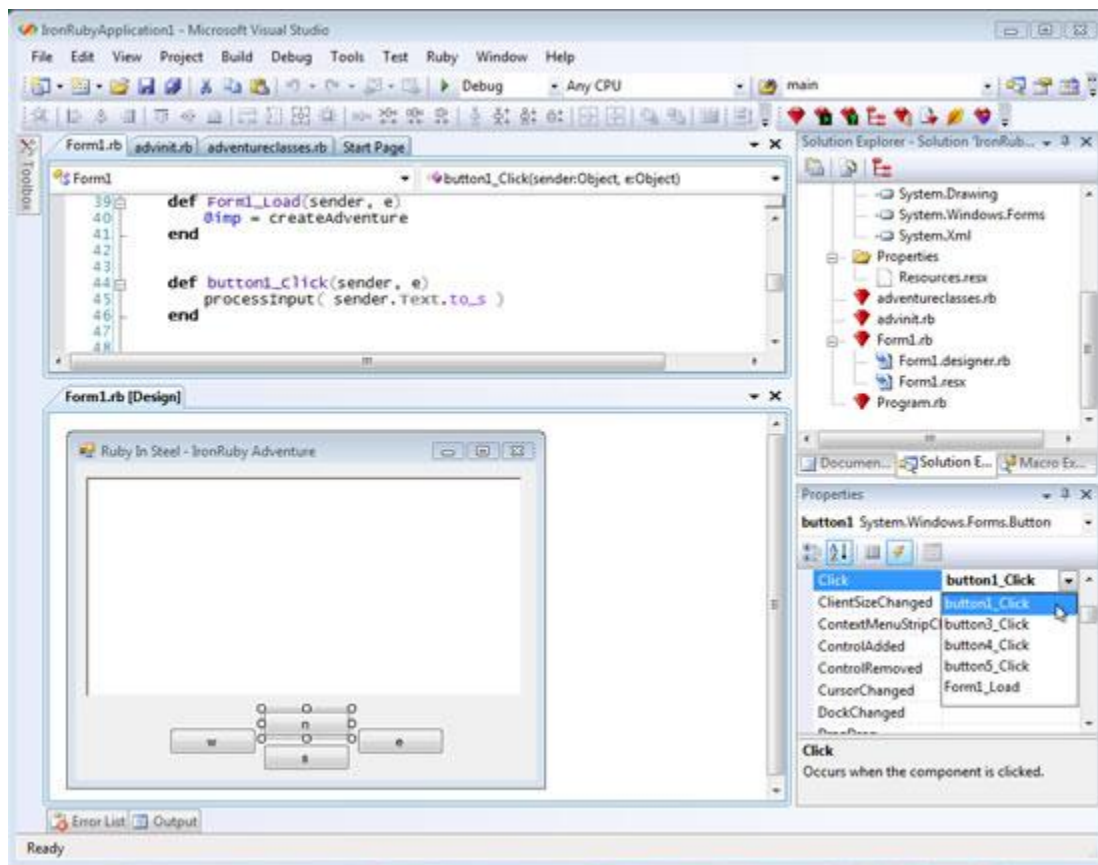
```
puts RUBY_PLATFORM
puts RUBY_VERSION
ENV_JAVA.each{ |e| puts( e ) }
```

RUBY\_PLATFORM should evaluate to 'java'. RUBY\_VERSION should show a number such as 1.8.6 and ENV\_JAVA should display numerous items relating to your Java installation.

If you wish to run Rails using JRuby, you will need to set up the JRuby On Rails system in advance by installing a number of gems. Refer to the JRuby site and Wiki for assistance.

## IronRuby Support

At the time of writing, Microsoft's IronRuby is still in development and, currently, it neither provides full compatibility with standard Ruby nor has all the features required for full and seamless integration with Visual Studio and .NET. Ruby In Steel's IronRuby support has 'alpha' status and should be regarded as provided for exploration of IronRuby rather than for application development. We shall upgrade our IronRuby support as the IronRuby project itself progresses. Our current support for IronRuby includes a drag and drop form designer (with some limitations), Ruby editing and project management.



## TO START AN IRONRUBY PROJECT.

- Select *File, New Project*, navigate to IronRuby project type branch and select *Iron Ruby Windows Application*.

Alternatively, you may select *IronRuby Console Application* in order to run text-mode applications in the docked Ruby Console. The steps below assume you have created a Windows application.

- Name the project and click OK.
- You will now see a blank form. Use the Toolbox to drop on components.  
e.g. Drop on a button (*button1*) and a textBox (*textBox1*).
- Double-click a control to generate an event-handler for its default event or select a named event from the Properties panel Events pane.  
e.g. Double-click *button1* to create this method in the code editor:

```
def button1_Click(sender, e)
end
```

- Edit the event-handler code.

e.g.

```
def button1_Click(sender, e)
    textBox1.Text = "Hello world"
end
```

- Press CTRL+F5 to run.

Please refer to the SapphireSteel web site for more information on IronRuby including details of any updates.

## Appendix

### The Ruby In Steel All-In-One-Installer

If you already have Visual Studio, Ruby and (optionally) Rails installed, you need only install Ruby In Steel itself. However, if you need to set up Ruby, Rails or a database (MySQL) Server - or if you don't own Visual Studio and wish to install a free Ruby-specific copy of Visual Studio 2008, you should use the All-in-one Installer. This lets you install some (or all) of the following:

- Ruby
- Rails
- Gems
- MySQL
- Visual Studio 2008 (free edition)
- .NET 3.5

The All-in-one Installer is documented in a separate PDF document supplied with the Ruby In Steel software: **Installation Instructions.pdf**.

The current installation packages for Ruby In Steel are available from the download page:

<http://www.sapphiresteel.com/spip?page=download>

### Setting Up Ruby and Rails

If you choose not to use the All-in-one Installer, you may install Ruby (and Rails) using other installers. Some options are described below...

### How To Install Ruby

The **Ruby One-Click Installer** provides a simple way to install Ruby along with some useful libraries. This may be obtained from: <http://www.ruby-lang.org/en/downloads>



## How To Install Rails

### INSTALL RUBYGEMS

First, make sure you have Gems installed. Gems is a package manager for Ruby which helps to install programs from disk or over the Internet. You will need to be connected to the Internet when installing Rails.

Your Ruby installation may already include Gems. To verify this, go to the command prompt (*Start Menu->Run (or Search)*), enter **cmd** and click OK). At the prompt enter: **gem**. If you see a screen of help information on RubyGems, all is well. If not, you need to install Gems. The download link and installation instructions can be found here:

<http://www.rubyonrails.org/down>.

Once Gems is installed, go to the command prompt and enter:

**gem install rails --include-dependencies**

RubyGems will now connect to the internet, download and install all the Rails libraries and utilities. Once it is complete you can verify the installation by entering at the prompt:

**rails -v**

This will display the version number of your Rails installation.

## How To Install MySQL

If you are working with Rails, you will need to install a database. While there are quite a few possible choices available to you, one of the most widely used is MySQL. You can download and install MySQL for free. However, if you've never used MySQL before, you may find some of the setup options confusing. Here, we'll try to guide you through the process to avoid potential problems...

The MySQL main site is at <http://www.mysql.com/> and from here you can navigate to the download page for the current version.

### DOWNLOAD MYSQL

We shall assume that you will be using the free edition of MySQL. This is available for download on the <http://dev.mysql.com/downloads/> page. The current version, at the time of writing, is *MySQL 5 Community Server*. The name and version number will, of course, change over time. Download whichever is the current (not upcoming, alpha or beta) release. Choose the specific version recommended for your operating system (there may be different versions for Win32 and Win64, for example).

You will need to scroll some way down this page to locate the Windows installers. You can either download the complete MySQL package or the smaller *Windows Essentials* package. The complete package contains extra tools for database developers but these are not required for simple Rails development. For most people, therefore, the smaller Windows Essentials download file is the one to get.

You should click the 'Pick A Mirror' link alongside this option. You will then be shown a questionnaire which you can fill out if you wish. If you don't wish to do so, just scroll down the page and pick a regional download site. Click a link and save the file, which will be named something like (the numbers may differ): **mysql-essential-5.0.41-win32.msi**, to any convenient directory on your disk.

### INSTALL MYSQL

Once the download has completed run the program by selecting Open or Run in the download dialog if this is still visible, or by double-clicking the installation file via Windows Explorer.

**Note:** During the installation of MySQL some advertising screens may appear. Click the buttons to move through these. Some security warnings may also prompt you to verify your intention to install the software. When prompted, you should click the necessary options to continue with the installation.

The first page of the Setup Wizard will now appear. Click the Next button.

You can either leave the *Typical setup* option selected if you are happy to install the software into the default MySQL directory beneath *C:\Program Files\*. If you want to install to some other directory, however, select *Custom*. Then click Next. Click *Change* to change the directory.

When you are ready to move on, click Next. You will see the screen stating ‘Ready To Install the Program’, verify that the destination folder is correct, then click the *Install* button.

Depending on the version of MySQL you may now either be shown some promotional screens or you may be prompted to create a new MySQL account which will let you receive news of changes and updates. These are not an essential part of the software installation and you may click the Next or Skip buttons to move on through the installation.

The Wizard Completed dialog now appears. Click the Finish button.

## **CONFIGURE MYSQL**

In fact, this isn’t the end of the installation after all. With some installers, a new screen pops up now welcoming you to the *MySQL Server Instance Configuration Wizard*. If this does not occur, you will need to load this yourself. Click the Start menu, then *Program->MySQL->MySQL Server 5.0* (or whichever version number you are using) then *MySQL Server Instance Config Wizard*. Click Next.

Assuming that this is the first time you’ve installed MySQL on this machine, you can select Standard Configuration (if you are upgrading from an older version of MySQL you need to select Detailed Configuration – that is beyond the scope of our simple setup guide). Click Next.

In the next dialog, leave the default options selected (i.e. *Install As Windows Service*; *Service Name* = ‘MySQL’ and *Launch the MySQL Server automatically*). Then click Next.

In the next screen, leave 'Modify Security Settings' checked and enter the same password (of your choice) into the first two text fields. You will need this password later so remember it or write it down in a secure location. If you may need to access MySQL from another computer you can check 'Enable root access from remote machines'. Then click Next.

**Note:** The default MySQL *user name* is '**root**'. The *password* is the one you just entered. You will need both these items of information later when creating Rails applications.

The next screen just gives you some information about the tasks that are about to be performed. Click the Execute button.

If you have previously installed or configured MySQL, you may see an error message which tells you to Skip the installation. You may click Retry to see if you can bypass this problem. If not, press Skip and then restart the MySQL configuration process, selecting *Reconfigure Instance* and *Standard Instance* when prompted.

When everything is installed this screen appears. Click Finish.

And that's it!

Just to test that everything's working, you can open the *MySQL command line client*. You can do this from the MySQL group on the Windows start menu. A 'DOS box' will appear and you will be prompted to enter your password. Once you've entered this, you will be welcomed to the MySQL monitor with a **mysql>** prompt. Enter *\h* for some help. Enter *quit* to exit.

## CONFIGURE MYSQL IN RUBY IN STEEL

Finally, you should check the paths to MySQL in Ruby In Steel. Load Visual Studio. Select *Tools->Options->Projects and Solutions->Ruby In Steel*. Verify that the MySQL Server path is set to the *\bin* directory into which MySQL Server was installed (e.g. *C:\Program Files\MySQL Server 5.0\bin*). You may browse to locate the install directory if necessary. Then click OK.

You should now be all set up to use MySQL with Ruby In Steel.

## More Information

### On Ruby In Steel

Visit the SapphireSteel Software web site for regularly updated information, hints and tips on using Ruby In Steel:

<http://www.sapphiresteel.com>

Or link straight to the Developers' Blog:

<http://www.sapphiresteel.com/-Blog->

We also have a discussion and support forum at:

<http://www.sapphiresteel.com/forum/>

### Tutorials and FAQ

If you have a technical problem, be sure to see if this has been answered on the FAQ:

<http://www.sapphiresteel.com/-FAQ->

We also have a number of tutorials:

<http://www.sapphiresteel.com/-Tutorials->

## **On Ruby**

If you are learning Ruby, you can download a free tutorial E-Book including all the source code, The Little Book Of Ruby:

<http://www.sapphiresteel.com/The-Little-Book-Of-Ruby>

Links to additional sources of lessons and documentation can be found on the Ruby Documentation site:

<http://www.ruby-doc.org/>

## **On Rails**

The primary source of online information on Rails is the Ruby On Rails web site:

<http://www.rubyonrails.org/>