


preamble/gfx/ruby.png

Mini Project in Partial Fulfillment of SW9, Fall 2006
Kristian Kristensen
Department of Computer Science
University of Aalborg

TITLE: The State of Coding Aid in Ruby IDE's
THEME: Mini Project
PROJECT PERIOD: SW9, Spring 2007
PROJECT GROUP:

GROUP MEMBERS:
Kristian Kristensen
kk@cs.aau.dk

SUPERVISOR:
Kurt Nørmark
normark@cs.aau.dk

Abstract

This report evaluates the state of Coding Aid in Ruby Integrated Development Environments (IDEs). First a definition of Coding Aid (aka IntelliSense and Content Assist) and its components (Method, Variable, Parameter, and Keyword Completion, Code Expansion and Inline Documentation) is provided. This is followed by a discussion of the problems involved in creating Coding Aid for a dynamically typed language like Ruby – a comparison between Smalltalk and Java is used. Via a set of code examples illustrating the problematic features of dynamic languages 9 IDEs are tested. The result is that Sapphire in Steel a plugin for Microsoft Visual Studio 2005 and IDEs utilizing rcode tools (TextMate, Vim, and Emacs) performs best.

COPIES: 2
PAGES: 25
WITH APPENDICES: 28
FINISHED: February 26, 2007

Preface

This report is written in partial fulfillment of my 9th semester, which was spent at West Virginia University. It surveys the state of Coding Aid (defined in Chapter 2 - Definition and Evaluation) in Ruby IDEs.

The following conventions are used throughout the report:

- The `mono-spaced font` is used to signify classes, methods, and namespaces.
- References are marked as [X] where X is a reference to the bibliography.

Source Code examples used in the report can be found online at <http://www.kristiankristensen.dk/sw9/>.

Kristian Kristensen

Contents

1	Introduction	1
2	Definition and Evaluation	3
2.1	The Problem with Coding Aid for Dynamically Typed Languages	4
2.2	Ruby Code Examples	7
2.3	Evaluating 9 Ruby IDEs	10
2.3.1	Feature Matrix	12
2.4	Discussion	15
2.5	Summary	16
3	Conclusion	17
	Bibliography	21
	Acronyms	23
A	Ruby Code Examples	25

Introduction

IntelliSense is used to help the programmer when working in an IDE. It suggests possible completions of the programmers input, limits what must be typed and lowers development time. In order to achieve this the IntelliSense feature must be integrated with the language used for development.

Ruby is a dynamically typed language, which is evaluated at runtime. This makes it difficult to create an IntelliSense feature for the language, because many aspects won't be known until runtime. The primary example of this are types, which won't be accurately known until the program is executed.

This project evaluates the IDEs available for Ruby programmers, and what type of IntelliSense they offer. Furthermore an evaluation will be done on the way the IDEs have implemented IntelliSense and overcome the inherent problems between this feature and dynamically typed languages.

The motivation for IntelliSense is manyfold. By its proponents it is said to encourage proper and descriptive method and variable names, because the programmer need not type them again; IntelliSense helps with that. It provides inline documentation by annotating code with information presented to the programmer such as what argument types a method takes and what it returns. And above all because it speeds up the development. However, it is also said to encourage sloppy coding practices, helping programmers know IntelliSense instead of the software libraries they use.

This report defines IntelliSense and its components, and performs an evaluation of IDEs for the programming language Ruby.

Definition and Evaluation

The term IntelliSense is usually used to denote a number of different features. Therefore this paragraph breaks the term into its constituents. Generally speaking IntelliSense deals with auto completion of whatever input a programmer gives his IDE. What differs between the sub-parts are the input upon which the IDE reacts and what the result of the operation is. Furthermore IntelliSense is continuously subject to the context in which it is being executed. Effectively this means that in order to be useful IntelliSense must take into account the input upon which it is being invoked, i.e. invoking IntelliSense using method completion on the input *a* should yield only the methods beginning with the letter “a”.

The feature in question is in industry colloquially known as IntelliSense coined by Microsoft Corp. [2007] or Code Assist by Eclipse Foundation [2007a]. Wikipedia [2007] describes the feature as “Auto Completion”¹ or “Code Completion”. In general it seems that some confusion exists on how to name this feature and what it consists of.

This chapter will define the features individual parts in relation to a programming environment. Furthermore a more neutral and less marketing-like term will be presented.

This feature and its sub-parts are implemented to aid the programmer in his work which is to produce code. Therefore it seems obvious to term it “Coding Aid”², and this is the term used for the rest of this report.

With an overall term for the feature the individual parts of it can be defined as the following.

¹“Auto Complete” also relates itself to uses other than strictly programming such as word processing.

²“Coding Aids” in plural.

Chapter 2. Definition and Evaluation

Method Completion: By some stimuli the programmer is presented with a list of possible method name completions filtered on the characters typed as part of the identifier.

Parameter Completion: When a valid method is detected, the programmer is presented with a visual indication of the parameters and their type. If the method is overloaded several choices are presented, however, if the method call is already fully typed with sets of parameters the valid one is highlighted.

Variable Completion: Based on the languages scoping and lexical rules a list is presented with valid variable name completions.

Code Expansion: Typing a set of predetermined “magic” characters triggers an action that replaces the characters with another block of code.

Keyword Completion: The programmer is presented with a list of the valid keywords of the language.

Inline Documentation: The code is annotated with documentation such as argument types, return types, overloads, etc. Often accomplished using tool tips based on the mouse hovering position.

The course of action is to define the problem of creating coding aid for dynamically typed languages, followed by a presentation of Ruby code examples that illustrates these difficulties in relation to the definitions presented above. Finally an evaluation of different Ruby IDEs capabilities on these code examples is performed. This report will focus on evaluating capabilities for method and variable completion.

2.1 The Problem with Coding Aid for Dynamically Typed Languages

In “A Summary of ‘Why is there no Smalltalk [or Java]-like IDE for Ruby’”, Joseph Moore [2007] summarizes the replies he got when posting that question on Usenet mailing lists. He presents two challenges that were brought up during the discussion:

- Language differences between Ruby and Smalltalk

- Language differences between Ruby and Java

The primary difference between Ruby and Smalltalk is that Smalltalk uses an *image based model*. The development environment for Smalltalk is Smalltalk and every change in the code is reflected directly in the image. The code is not re-evaluated at every execution (as in Ruby), rather the image is saved between invocations and hence evolves over time with the program. This implies that the Smalltalk environment always knows what have happened in the program, and it adds extensive methods for querying this meta data. As noted Ruby evaluates the program every time it is run, i.e. it doesn't save information or meta data between runs, which could be used for information mining.

The difference between Ruby and Java is straight forward: Java is statically typed, Ruby is dynamically (*loosely*) typed, Java is compiled, Ruby interpreted. Java knows every type of every object in the code, because it is defined as part of the language – `List x` – whereas the types in Ruby are deduced at run time.

Joseph Moore [2007] concludes his summary by suggesting that a Ruby development environment be implemented that includes some of Smalltalk's ideas; namely to incorporate some kind of memory between executions of the code to enable better meta data.

Providing Coding Aid as defined above all boils down to knowing the type of objects in the code. With the type the information needed to present completion lists becomes available. Therefore a sub problem of Coding Aid is to do type inference on dynamically typed languages.

Madsen and Soerensen [2006] implemented a type inference system for Ruby in Java and concludes "*The greatest challenge in doing type inference for Ruby was to model the semantic of the language.*" [Madsen and Soerensen, 2006, page 61]. Their system is fairly complete, however, it doesn't work on partial code, i.e. the code must be syntactically correct for their implementation to work. Nevertheless they exemplify that it is possible to create do type inference on Ruby code. This result is crucial to create Coding Aid for Ruby code.

Collingbourne [2006b], who co-developed Sapphire in Steel a Ruby plugin for Microsoft Visual Studio, writes "*In order to get IntelliSense right – that is, a system which produces completion lists that show the correct methods and variables according to the scope and type of any given object – our software is constantly analysing code as it is typed. That's a hard problem to deal with even in a statically typed language. With a language as dynamic as Ruby – one that not only*

Chapter 2. Definition and Evaluation

doesn't have type declarations but also allows you to write code that works with various types of object, only determined at runtime – this problem becomes phenomenally difficult: indeed, in many circumstances, there is no 'correct' solution to the problem. ... We can't succeed 100% in this; we can try – but the problem is inherently insoluble.". In one of the many posts on the Sapphire in Steel development team blog, they exemplify this claim with the following code [Sapphire In Steel, 2006]:

```
1 # http://www.sapphiresteel.com/IntelliSense-in-depth
2 puts "randomizing x"
3
4 x = case rand(3)
5   when 1
6     Array.new
7   when 2
8     Hash.new
9   when 3
10    "hello"
11  else
12    true
13  end
14 puts "x is: " + x.to_s
```

No matter what analysis is done on the code, the actual type of x will not be explicitly determined until runtime. The analyzer can make reasonable deductions about the possible types of x and possibly combine this information to the programmers benefit.

Another question to ask is how often does one see code like that even though it is allowed in a dynamically typed language? Real life code will probably exercise certain common sense characteristics, if not for another reason than to keep the programmer sane.

The problem with Ruby/dynamically typed languages over static ones like C# is that the state of the code is linked to the execution of the code. Ruby code can be self modifying and the state of it is intrinsically linked to the actual interpretation or execution of it. In Table 2.1 this is exemplified by having code that reopens a class and adds a method to it. In programming languages like C# and Java the state of the code is fixed, the execution environment changes. In Ruby the code state is fluid, as well as the execution environment. This is in contrast to the Smalltalk language and its development environment.

3

³**FiXme:** *There is a problem with the code. Check it!*

```
1 class Test
2   def m1()
3     puts "m1"
4   end
5 end
6
7 a = Test.new
8 a.m1
9 # m1 exists here, a. will yield m1 in list
10 # a.m2 will result in NoMethodError
11
12 class << Test
13   def m2()
14     puts "m2"
15   end
16 end
17 a = Test.new
18 #puts a.methods
19 a.m2
20
21 #m2 exists here, a. will yield m1 and m2 here.
```

Table 2.1: A Ruby class which is reopened and a method added to it. Depending on the position in the code one or both methods are defined.

2.2 Ruby Code Examples

This section evaluates a number of language aspects that makes a Coding Aid engine for Ruby difficult to produce. For each language aspect a piece of example code is presented that illustrates the problem. These code examples are used as the basis for the evaluation of the IDE.

Variables in Ruby – as in most other programming languages – have scope, they are defined and usable in certain contexts. The code below illustrates different ways to declare and use class, instance, and local variables.

```
1 # Variable scope
2 class Test
3   # @@ denotes class variable
4   @@varClass = "class"
5
6
7   def initialize
8     # @ denotes instance variable
9     @varInstance = "instance"
10  end
11
12  def my
13    # no prefix denotes local variable
14    varLocal = "local"
15    @varInstance
16  end
17 end
18
```

Chapter 2. Definition and Evaluation

```
19 puts "hello world"
20 x = Test.new
21 puts x.my
```

Ruby being a dynamically typed languages allows a variable to assume many different types throughout its lifetime. The below, which was mentioned above, illustrates this.

```
1 # http://www.sapphiresteel.com/IntelliSense-in-depth
2 puts "randomizing x"
3
4 x = case rand(3)
5   when 1
6     Array.new
7   when 2
8     Hash.new
9   when 3
10    "hello"
11  else
12    true
13  end
14 puts "x is: " + x.to_s
```

Ruby doesn't include statements, everything is an expression and evaluates to a value. A prime example of this is the code seen below. The "if" evaluates to a string, and can therefore be called with string methods.

```
1 # If expression
2 x = (if true
3     "hello world"
4     end
5     ).capitalize
6 puts x
```

A feature of Ruby is that instance variables need not be defined in a specific place. You can opt to define them initially in your class, or you can define it in an instance method when you need it. The problem for the Coding Aid engine then becomes: how do you collectively gather all the instance variables defined?

```
1 # Instance vars
2 class Test
3   @topLevelVar = "hello"
4
5   def myMethod
6     @newInstanceVar = "world"
7   end
8 end
```

Mixins are Ruby's way of tackling multiple inheritance or programming via interfaces traditionally found in C# and Java. Mixins are included in classes and augment the latters functionality.

```
1 # Mixins
```

```
2 module WhatsYourName
3   def myNameIs
4     "Test"
5   end
6 end
7
8 class MyClass
9   include WhatsYourName
10
11 end
12
13 m = MyClass.new
14 puts m.myNameIs
```

As mentioned above and seen in Table 2.1 Ruby allows for reopening of classes and adding behavior to them. The question for the Coding Aid engine is to figure out when a specific type assigned to a variable includes which functionality at what point in time/code.

A high-praised feature of Ruby is its dynamic or meta-programming facilities that allow the programmer to change almost anything on run-time including adding and removing methods. A prime example of this are the built-in methods *attr_reader*, *attr_writer*, and *attr_accessor*. They take a symbol⁴ as argument and generate attribute readers, writers or both respectively for an instance variable represented by the symbol argument. Effectively the three methods inject the appropriate methods into the class defining them. The measures for doing so are full available for the Ruby programmer to take advantage of.

The problem for the Coding Aid engine is to know what these methods do, because they use the dynamic features of Ruby. A simple measure is to add knowledge of these three special methods to the engine, but this solution doesn't scale. The method *class_eval* in combination with "here documents" presents a particular set of problems for a Coding Aid engine. The code below defines a class, and uses *class_eval* to add a method (*myInjectedMethod*) to all instances of the class Test.

```
1 # Example adapted from http://steve.yegge.googlepages.com/digging-into-ruby
  -symbols
2 class Test
3   def initialize
4
5   end
6 end
7
8 Test.class_eval <<CODE
9   def myInjectedMethod()
10    puts 'Hello World'
11  end
12 CODE
```

⁴Collingbourne [2006a] defines "In Ruby, a symbol is a name preceded by a colon. Symbol is defined in the Ruby class library to represent names inside the Ruby interpreter."

Chapter 2. Definition and Evaluation

```
13
14 x = Test.new
15 puts x.myInjectedMethod
```

The problem for the Coding Aid engine is obvious: how is it to know what the result of running *class_eval* or other variants of *eval*?

The list of tests to run on each IDE is:

Variable Scopes (VS): Coding Aid is provided for variables in their correct context.

Variable with Several Types (VST): A variable assumes a different type depending on execution.

Expressions (E): The result of an if expression

Defining Instance Variables (DEI): Instance variables are included in Coding Aid.

Mixins (M): Coding Aid is supplied for mixed in modules.

Reopening Classes (RC): Coding Aid is done with respect to reopening classes.

Attributes (A): Coding Aid is provided when the three attribute dynamic methods are used.

Class Eval (CE): The injected method is included in Coding Aid.

2.3 Evaluating 9 Ruby IDEs

Searching for Ruby IDEs yields quite a long list. Most of these applications claim to be the one and only solution for your Ruby development needs. This following will serve to answer if that is true when it comes to coding aid.

A set of criteria will be set forth that is used to classify each IDEs. The classifications are as follows:

IDEs Type: Values can be editor, specialized towards one purpose/language, multipurpose for use with different languages/purposes.

Completion type: Ctags based, custom developed code analysis (Anal.), hooking into Ruby compiler (RCT).

Platform: Windows, Mac OS, Linux

Furthermore each IDEs will be tested with a set of code examples to verify the degree of coding aid they support.

The applications included in the evaluation are selected based on their public ubiquity and to cover the field of applications. The applications selected are:

Eclipse [Eclipse Foundation, 2007b] with Ruby Development Tools (RDT) [RDT Project Team, 2007]:

A plug-in for the Eclipse Platform to provide Ruby support. Based on work done by Jason Morrison via Google Summer of Code, they are adding coding aid to RDT.

Another popular option for doing Rails⁵ development is RadRAILS, which repackages Eclipse, RDT and more into one package aimed for Ruby on Rails development. However, since it is basically Eclipse and RDT it is not included separately.

Microsoft Visual Studio 2005 with Sapphire in Steel [Hogan and Collingbourne, 2007]:

Steel expands Visual Studio 2005 to be a full-fledged application for Ruby development. The people behind have created IntelliSense, Debugging, and more for Ruby in Microsoft Visual Studio (VS).

jEdit [Pestov and jEdit Core, 2007] with Ruby Editor Plugin [McKinnon, 2007]:

Adds Ruby programming capabilities to jEdit – an editor.

FreeRIDE [FreeRIDE Team, 2007]: Aims to be THE IDE of choice for Ruby programming. Written entirely in Ruby it has high goals such as being cross-platform and supplying advanced functionality such as coding aid.

Arachno Ruby IDE [Scriptolutions, 2006]: Made by Scriptolutions it serves to create an editor for Ruby that merges some of the key ideas found in Emacs, Vi and Borland Delphi.

TextMate [Odgaard, 2007] with the Ruby Bundle: The ubiquitous editor for the Mac made famous in the Ruby community by David Heinemeier Hansson author of Ruby on Rails a web application framework. The Ruby Bundle adds Ruby support to this application.

⁵Ruby on Rails is a Web Development Framework: <http://www.rubyonrails.org>

Chapter 2. Definition and Evaluation

Vim [Vim Team, 2007]: A text editor with a long history. Supports numerous languages and modes and also Ruby.

Emacs[Free Software Foundation, 2007]: Text editor, platform or Operating System (OS) are just some of the words Emacs lovers associate with their editing application of choice. Emacs supports Ruby primarily via Ruby Mode.

ActiveState Komodo IDE 4.0 for Ruby [ActiveState, 2007]: Serves to be an IDE for dynamic languages and hence supports Ruby. It has a heavy focus on web development and is built on the Mozilla platform.

The following section discusses presents the results obtained after performing the test code evaluation in each application.

2.3.1 Feature Matrix

The results of performing the tests in each application are seen in Table 2.2.

Eclipse (RDT)

RDT Project Team [b] writes *“If you press Ctrl+space in the ruby editor, a list with information from the current ruby file is displayed. This information includes classes, modules, globals, methods and variables. There are also suggestions for keywords and some pre-defined globals. The keywords and globals are only shown if you have already typed at least one character. If the 'token' preceding the cursor does match the beginning of any suggestions you will get no suggestions.”*

FreeRIDE

As of September 2006 FreeRIDE does not include any Coding Aid feature [Maasland, 2006].

The State of IntelliSense in Ruby IDE's

IDE	Type	Impl.	Platform	VS	VST	E	DEI	M	RC	A	CE
RDT	IDE	Anal.	All	-	-	-	-	-	-	-	-
Sapphire	IDE	Anal.	Windows	+	+	+ ^a	+	+	- ^b	+	-
jEdit	Editor, multi	Anal.	All	+	- ^c	-	+ ^d	- ^e	-	-	-
FreeRIDE	Editor	None	All	-	-	-	-	-	-	-	-
Arachno	IDE	?	Win, Linux	+ ^f	-	-	+ ^g	-	+	-	-
TextMate	Editor	RCT	Mac OS	+ ^h	+	+ ⁱ	-	+	-	+	+
Vim ^j	Editor, multi	RCT	All	+	+	+	-	+	-	+	+
Emacs ^k	Editor, multi	RCT	All	+	+	+	-	+	-	+	+
Komodo	IDE	Anal.	All	+ ^l	- ^m	-	-	+	- ⁿ	+	-

^aAfter expression type is known, but () doesn't work

^bMight be wrong code

^cInfers the string type, but changing it to an integer doesn't change in the system

^dDisplays both, looks like it searches for identifiers

^eCan't infer x, so list is empty

^fClass vars doesn't work

^gScans file for identifiers

^hCan't recognize instance and class variables.

ⁱDoesn't work after parenthesis, but works after assignment to x.

^jResults for Vim are equivalent to those of TextMate

^kResults for Emacs are equivalent to those of TextMate

^lWorks on local vars, not on instance or class vars

^mFirst type assignment is kept

ⁿAdds the extra method to the class, but not with respect to defining location

Table 2.2: Results of running the tests in each IDE.

Arachno Ruby IDE

Arachno seems to parse the current file for valid identifiers and then uses this information to create completion lists. If a token has been entered it will be used to filter the list. This also explains why it finds all the instance variables defined in a class: it simply scans the file, and every identifier beginning with a is an instance variable. However, class variables doesn't work, which seems a bit odd considering the approach.

Sapphire in Steel

Sapphire in Steel uses a custom built Ruby parser that builds an Abstract Syntax Tree (AST) [Sapphire In Steel, b]. Every parse of the Ruby code is saved in a

Chapter 2. Definition and Evaluation

symbol table that is kept[Sapphire In Steel, e]. When a new parse has been made the old symbol table is copied and saved. This enables Coding Aid even though a new parse encounters a parse error. In the case where their tool can't infer the type of a method (argument types and return types) it is possible to annotate the code with the types, which the inference engine will pick up[Sapphire In Steel, c].

Dermot Hogan and Huw Collingbourne – the team behind Sapphire in Steel – have blogged about their experiences in implementing Coding Aid for Ruby [Sapphire In Steel, d,b,e,f, 2006, g,a,c; Collingbourne, 2006b; Sapphire In Steel, h]

jEdit

The Ruby Plugin for jEdit uses the SideKick support plugin to supply Coding Aid. SideKick is a general way of providing completions and display the structure of the editors files. The plugin writer creates a parser and SideKick delegates the requests for completions.

From the coding tests done with the plugin it seems that the majority of the Coding Aid is limited to the built in library of Ruby and the Standard Library. User defined classes doesn't seem to get properly recognized, and hence Coding Aid for them is not provided. The Coding Aid provided seems to be a list of identifiers found possibly filtered by what the programmer enters.

ActiveState Komodo

ActiveState writes in their documentation that they support “*Autocomplete for require statements...Autocomplete for available attributes on a class or module namespace ... Autocomplete for methods on an class instance... Calltips for method calls*” [ActiveState [b]. These statements corresponds approximately to the results found in the tests.

Furthermore, they support the generation of a “Code Intelligence Database” which is used to query against when supplying Coding Aid. However, they note that utilizing these databases are not required, they only provide a performance gain [ActiveState, a].

TextMate, Vim, Emacs

Ruby support in TextMate is handled via the Ruby Bundle⁶, and this description is therefore based on that and its capabilities. The latest version of the bundle uses a tool called *rcodetools* [Fernandez and rubikitch, 2007] for amongst other things coding aid. This package includes a command `rct-complete` which takes a Ruby code file and a line and column number for where to supply a list of valid completions. The package evaluates/executes the code, and uses built in Ruby methods to extract the valid completions at the specified place.

How this evaluation is accomplished without side effects is a bit unclear. The source code is available but the system isn't easily accessible.

Both Vim and Emacs is supported by *rcodetools*, which currently seems to be the state of the art for code completion in text based editors. The results of running the code examples are therefore equivalent to those of TextMate and the Ruby bundle.

2.4 Discussion

It seems that the two applications that offer the best method and variable completion are Sapphire in Steel and the applications utilizing *rcodetools* (TextMate, Vim, and Emacs). Sapphire in Steel supports the definition of instance variables, which *rcodetools* doesn't. However, *rcodetools* supports dynamic code or meta programming in the form of *class_eval*.

Sapphire in Steel are exclusively for Windows, whereas *rcodetools* is supported on all platforms – by Emacs and Vim and its only requirement a Ruby environment.

A set of the applications tested use their own analysis of Ruby code to create coding aid. However, gathering enough information to accomplish the task requires a large amount of work as described by Huw Collingbourne and Dermod Hogan the team behind Sapphire in Steel (see Section 2.3.1 - Sapphire in Steel).

The approach taken by *rcodetools* – evaluating the Ruby code and querying the environment for information – resembles the approach taken by Smalltalkers (see Section 2.1 - The Problem with Coding Aid for Dynamically Typed Languages). The advantage is that the evaluator of the code is also the one queried for infor-

⁶Bundles in TextMate lingo is an extension that enables TextMate to understand a particular language or environment.

Chapter 2. Definition and Evaluation

mation. Hence, correct information will be available. This method is not 100% comparable to Smalltalk, because the latter keeps an image of the code available between runs, whereas rcodetools will evaluate the code every time completion is requested.

Also interesting is the result that Eclipse lacks any completion for Ruby code, despite its efforts in RDT. The RDT web site [RDT Project Team, a] states that a Jason Morrison is working on type inference for Ruby to provide proper coding aid for Ruby code. However, no information as to his progress is available.

2.5 Summary

This chapter has accomplished a list of things. First the problem of providing coding aid for a dynamically typed language like Ruby was discussed. This resulted in a set of Ruby code examples that exercised these difficult parts. A list of Ruby IDEs were presented and they were evaluated using the code examples. A discussion of the results have been provided, which concludes that the IDEs with the best method and variable completion are Sapphire in Steel (a plugin for Microsoft Visual Studio 2005) and TextMate, Vim and Emacs with the help of rcodetools.

Conclusion

Conclude here

Chapter 3. Conclusion

Bibliography

- ActiveState. Code intelligence. <http://aspn.activestate.com/ASP/ docs/ Komodo/4.0/komodo-doc-codeintel.html>.
- ActiveState. Ruby autocomple and calltips. http://aspn.activestate.com/ ASP/ docs/ Komodo/4.0/komodo-doc-editor.html#Ruby_AutoComplete.
- ActiveState (2007). Activestate komodo ide 4.0. http://www.activestate. com/products/komodo_ide/.
- Collingbourne, H. (2006a). *The Little Book Of Ruby*. Rosedown Mill Ltd., first edition. <http://www.sapphiresteel.com>.
- Collingbourne, H. (2006b). Ruby variable completion. Sapphire In Steel <http://www.sapphiresteel.com/Ruby-Variable-Completion-The>.
- Eclipse Foundation (2007a). Content assist. [help.eclipse.org/ help31/topic/org.eclipse.platform.doc.isv/guide/editors_ contentassist.htm](http://help.eclipse.org/help31/topic/org.eclipse.platform.doc.isv/guide/editors_contentassist.htm).
- Eclipse Foundation (2007b). Eclipse - an open development platform. <http://www.eclipse.org>.
- Fernandez, M. and rubikitch (2007). rcodetools. [http://eigenclass.org/ hiki.rb?rcodetools](http://eigenclass.org/hiki.rb?rcodetools).
- Free Software Foundation (2007). Gnu emacs. [http://www.gnu.org/ software/emacs/](http://www.gnu.org/software/emacs/).
- FreeRIDE Team (2007). Freeride. <http://freeride.rubyforge.org>.

BIBLIOGRAPHY

- Hogan, D. and Collingbourne, H. (2007). Ruby ide :: Ruby in steel :: Ruby programming with visual studio 2005. www.sapphiresteel.com/.
- Joseph Moore (2007). A summary of "why is there no smalltalk [or java]-like ide for ruby". Blog. <http://40withhegg.com/2007/1/5/a-summary-of-why-is-there-no-smalltalk-or-java-like-ide-for-ruby>.
- Maasland, J. (2006). [fr-users] intellisense? Maillinglist. <http://rubyforge.org/pipermail/freeride-users/2006-September/000200.html>.
- Madsen, M. and Soerensen, P. (2006). Type inference in ruby. Technical report, Department of Computer Science, Aalborg University.
- McKinnon, R. (2007). jedit ruby editor plugin. <http://rubyjedit.org/>.
- Microsoft Corp. (2007). Visual studio - using intellisense. Web Site. [http://msdn2.microsoft.com/en-us/library/hcw1s69b\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/hcw1s69b(VS.80).aspx).
- Odgaard, A. (2007). Textmate. <http://www.macromates.com>.
- Pestov, S. and jEdit Core (2007). jedit. <http://www.jedit.org>.
- RDT Project Team. Rdt - ruby development tools. <http://rubyclipse.sourceforge.net/>.
- RDT Project Team. Ruby development tools documentation. Website. <http://download.rubypeople.org/release/0.8.0.604272100PRD/doc/html/ch02s04.html#CodeCompletion>.
- Sapphire In Steel. <http://www.sapphiresteel.com/Ruby-IntelliSense-scoping-fun>.
- Sapphire In Steel. The devil is in the detail. <http://www.sapphiresteel.com/The-devil-is-in-the-detail>.
- Sapphire In Steel. Filling out ruby intellisense. <http://www.sapphiresteel.com/Filling-out-Ruby-IntelliSense>.
- Sapphire In Steel. Intellisense and parsing ruby. <http://www.sapphiresteel.com/IntelliSense-and-Parsing-Ruby>.
- Sapphire In Steel. Intellisense phase 1. <http://www.sapphiresteel.com/IntelliSense-phase-1>.
- Sapphire In Steel. Quickinfo and rdoc. <http://www.sapphiresteel.com/QuickInfo-and-RDoc>.

The State of IntelliSense in Ruby IDE's

Sapphire In Steel. Ruby intellisense - the finer points. <http://www.sapphiresteel.com/Ruby-IntelliSense-the-finer-points>.

Sapphire In Steel. Ruby on rails intellisense. <http://www.sapphiresteel.com/Ruby-On-Rails-IntelliSense>.

Sapphire In Steel (2006). Intellisense in depth. **Blog.** <http://www.sapphiresteel.com/IntelliSense-in-depth>.

Scriptolutions (2006). Arachno ruby editor. http://www.ruby-ide.com/ruby/ruby_ide_and_ruby_editor.php.

Vim Team (2007). Vim. <http://www.vim.org>.

Wikipedia (2007). Autocomplete. <http://en.wikipedia.org/wiki/Autocomplete>.

BIBLIOGRAPHY

Acronyms

AST Abstract Syntax Tree

IDE Integrated Development Environment

OS Operating System

RDT Ruby Development Tools

VS Microsoft Visual Studio

Ruby Code Examples

```
1 # http://www.sapphiresteel.com/IntelliSense-in-depth
2 puts "randomizing x"
3
4 x = case rand(3)
5     when 1
6         Array.new
7     when 2
8         Hash.new
9     when 3
10        "hello"
11     else
12        true
13     end
14 puts "x is: " + x.to_s
```

```
1 # http://www.sapphiresteel.com/IntelliSense-in-depth
2 module YAML
3     module BaseNode
4         def b1; puts "hello from b1" end
5         def b2; puts "hello from b2" end
6     end
7
8     module Syck
9         class Node
10            include YAML::BaseNode
11
12            def self.xx; puts "there" end
13
14            class InnerNode
15                def in1; puts "hello again" end
16                def self.in2; puts "there again" end
17            end
18        end
19    end
20 end
21 end
```

```
1 # Variable scope
2 class Test
3     # @@ denotes class variable
4     @@varClass = "class"
5
6
```

Chapter A. Ruby Code Examples

```
7  def initialize
8    # @ denotes instance variable
9    @varInstance = "instance"
10 end
11
12 def my
13   # no prefix denotes local variable
14   varLocal = "local"
15   @varInstance
16 end
17 end
18
19 puts "hello world"
20 x = Test.new
21 puts x.my

1  class Test
2    attr_reader :name
3
4    def initialize
5      @name = "hej"
6    end
7  end
8
9  x = Test.new
10 puts x.name

1  # Instance vars
2  class Test
3    @topLevelVar = "hello"
4
5    def myMethod
6      @newInstanceVar = "world"
7    end
8  end

1  x = "Hello world"
2
3  x.capitalize
4
5  x = 1

1  # Example adapted from http://steve.yegge.googlepages.com/digging-into-ruby-symbols
2  class Test
3    def initialize
4
5    end
6  end
7
8  Test.class_eval <<CODE
9    def myInjectedMethod()
10     puts 'Hello World'
11   end
12 CODE
13
14 x = Test.new
15 puts x.myInjectedMethod

1  # Mixins
2  module WhatsYourName
```

The State of IntelliSense in Ruby IDE's

```
3   def myNameIs
4     "Test"
5   end
6 end
7
8 class MyClass
9   include WhatsYourName
10
11 end
12
13 m = MyClass.new
14 puts m.myNameIs

1 # http://www.sapphiresteel.com/IntelliSense-in-depth
2 puts "randomizing x"
3
4 x = case rand(3)
5     when 1
6       Array.new
7     when 2
8       Hash.new
9     when 3
10      "hello"
11     else
12      true
13     end
14 puts "x is: " + x.to_s

1 # If expression
2 x = (if true
3     "hello world"
4     end
5     ).capitalize
6 puts x

1 class Test
2   def m1()
3     puts "m1"
4   end
5 end
6
7 a = Test.new
8 a.m1
9 # m1 exists here, a. will yield m1 in list
10 # a.m2 will result in NoMethodError
11
12 class << Test
13   def m2()
14     puts "m2"
15   end
16 end
17 a = Test.new
18 #puts a.methods
19 a.m2
20
21 #m2 exists here, a. will yield m1 and m2 here.

1 # http://www.sapphiresteel.com/IntelliSense-in-depth
2 module YAML
3   module BaseNode
4     def b1; puts "hello from b1" end
5   end
6 end
```

Chapter A. Ruby Code Examples

```
5     def b2; puts "hello from b2" end
6   end
7
8   module Syck
9     class Node
10      include YAML::BaseNode
11
12      def self.xx; puts "there" end
13
14      class InnerNode
15        def in1; puts "hello again" end
16        def self.in2; puts "there again" end
17
18      end
19    end
20  end
21 end
```