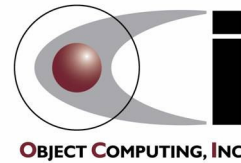


Ruby Tools

No Fluff Just Stuff
Gateway Software Symposium
March 3-5, 2006

presented by
Mark Volkmann



Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Talk Contents

So you're ready to try Ruby, but not yet familiar with tools for working with Ruby? This talk will get you started with using these tools, many of which are provided with Ruby and others which are also freely available.

1 Basic Tools

- ruby interpreter
- irb
- ri

2 Unit Testing

3 Advanced Tools

- IDEs and editors
- RDoc
- Logging
- Debugging
- Profiling

4 RubyGems

5 SWIG

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

2

Ruby Interpreter

- **ruby** [*options*] [*script-name*] [*arguments*]

- options

- c only checks syntax
- d sets \$DEBUG global variable to true
- e '*statement*' for one-liners
example: `ruby -e 'puts Time.now'`
- h outputs help on these options
- n executes code in –e on each line in an input file (stored in \$_) ←
- p same as –n, but also print each line after processing it
- r*library* requires a given library before running script
- v outputs version number
- w outputs warning messages

↑
stored in the
global array **ARGV**,
synonym for \$*

–n example

The file `cars.txt` contains
1997 Saturn SC2 purple
1999 Pontiac Firebird black
2001 Honda Odyssey green
2001 BMW Z3 yellow

To print the car colors use
`ruby -n -e "puts $_.split[3]" cars.txt`

Ruby Interpreter (Cont'd)

- **rubyw**

- Windows-only
- just like **ruby**, but doesn't open a window when run from a script
- useful for double-clickable GUI applications

- **RUBYOPT** environment variable

- contains a space-separated list of options used by `ruby`, `rubyw` and `irb`
- if only one option is specified, the dash can be omitted
- examples

- "`rubygems`" requires "`ubygems.rb`" more on this in the RubyGems section
- "`-rubygems -w`" requires "`ubygems.rb`" and outputs warning messages

1

Interactive Ruby - irb

- Useful for interactively experimenting with code
- Steps to use
 - open a shell window or command prompt
 - enter `irb [options]`
 - enter any sequence of Ruby statements
 - to read Ruby statements from a file
 - `require 'file-name'`
 - only loads the first time
 - can load both Ruby source files and binary shared libraries
 - `load 'file-name.rb'`
 - loads every time
 - useful in irb to reload a modified file
- To exit
 - enter `exit` or `quit`

`require` and `load` search for files in the load path stored in the global variable `$LOAD_PATH`, which is a synonym for `$:`

1

irb Command-line Options

- Command-line options
 - `-h` outputs help on irb
 - `-d` sets `$DEBUG` global variable to true
 - `-I dir` adds a directory to `$LOAD_PATH`
can specify any number of these
 - `-r file-path` reads Ruby statements from given file
 - `-v` outputs version number of irb

1

irb Configuration

- **Configuration**
 - many configuration options are available (see pickaxe 192)
 - enter `conf` in irb to see current settings
 - can set within irb or specify in configuration file
 - named `.irbrc`, `irb.rc` or `_irbrc`
 - searches for file in `~` (Linux) or `C:/Documents and Settings/user-name` (Windows)
 - for example, enable auto-indent to indent code as it is entered
 - within irb, enter `conf.auto_indent_mode = true`
 - in `.irbrc`, add `IRB.conf[:AUTO_INDENT] = true`
- **Tab completion**
 - may be enabled by default
 - if not, enter `require 'irb/completion'` ← can add this to your irb configuration file
 - works for class, module and method names
 - if more than one match, options are displayed when tab key is pressed a second time

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

7

1

Example irb Session

```
C:\Ruby>irb

irb(main):001:0> s = 'Programming Ruby'
=> "Programming Ruby" ← outputs inspect value of
                        each expression entered

irb(main):002:0> require 'irb/completion'

irb(main):003:0> s.u ← press tab twice for list of completion
                    matches since there is more than one
s.unpack s.untaint s.upcase s.upcase! s.upto

irb(main):003:0> s.upcase
=> "PROGRAMMING RUBY"

irb(main):004:0> s.methods.grep /case/ ← get list of String methods
                                        whose name contains "case"
=> ["upcase", "upcase!", "downcase", "downcase!", "casecmp",
    "swapcase", "swapcase!"]

irb(main):005:0> exit

C:\Ruby>
```

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

8

ri

- **Command-line tool that displays documentation on classes, modules and methods**
 - for now, only built-ins and some standard libraries
 - determined by `.document` files below Ruby installation
- **Usage**
 - **ri -h**
 - outputs help on using ri
 - **ri name**
 - outputs documentation on the given name
 - can be a partial name
 - can be a name of a class, module or method
 - for methods in more than one class, displays a list of matches
 - can be `class.method` or `module.method`
 - **ri -c**
 - outputs a list of classes and modules for which documentation is available

also consider **fxri** which is a GUI that combines ri and irb; included by Windows One-Click Installer

For example, several classes have a method named "succ". To see documentation for the one in the Date class, enter "ri Date.succ".

ri (Cont'd)

- **Class/module methods**
 - in Ruby, these can be invoked with `module-name::method-name` or `module-name.method-name`
 - can use either `::` or `.` to search ri
- **Instance methods**
 - in Ruby, these must be invoked with `object.method-name`
 - can use either `#` or `.` to search ri
 - `#` is just a documentation convention for instance methods
- **Uses "less" to display**
 - for help on keystrokes, type `'h'` (similar to vi)
 - `f` or `PgDn` scrolls forward, `b` or `PgUp` scrolls backward
 - `g` goes to first line, `G` goes to last line
 - `/pattern` searches forward, `n` repeats last search forward, `N` repeats last search backward
 - `q` exits

Unit Testing Overview

- **What to test?**
 - everything that could possibly go wrong
 - open to interpretation; test accessor methods?
- **With a good suite of tests ...**
 - there is less chance of introducing bugs when code is modified or refactored
- **“unit testing”?**
 - tests often go beyond the scope of a single “unit”
 - better term might be “automated testing”
 - don't want to have to examine detailed output to determine if all tests passed

Testing Strategies

- **Write tests before code (test-first)**
 - seen by most as the **best strategy**, even by those that don't do it
 - forces thinking about what is really needed
 - the tests become a kind of requirements documentation
 - helps with API design
 - stop writing code when all tests pass
- **Write a little, test a little, repeat**
 - **most common strategy** because many developers feel writing code is more fun than writing tests
- **Write all the code, then all the tests**
 - **worst strategy**
 - much of the benefit of having the tests is lost, but still far better than having no tests

Ruby Unit Testing

- Ruby comes with `Test::Unit` framework
 - a standard library
- Steps to use
 - `require 'test/unit'`
 - create class that inherits from `Test::Unit::TestCase`
 - write methods whose names begin with “test”
 - add “assert” calls to those methods
 - put code to be run before each test method in `setup` method
 - put code to be run after each test method in `teardown` method

Placing and Running Tests

- Recommended project directory structure
 - *project-dir*
 - README
 - `bin` directory - for shell scripts
 - `doc` directory - for documentation
 - `lib` directory - for Ruby project source files (*.rb)
 - `ex.Foo.rb`
 - `test` directory - for Ruby unit test source files (*Test.rb)
 - `ex.FooTest.rb`
- Running tests
 - run tests from `lib` directory
 - `$RUBY_HOME/bin` should be in `PATH`
 - `testrb` script is located there
 - run with `testrb [-rtk] ../test/*.rb`
 - by default, outputs results to console
 - “-rtk” option outputs results in a Tk GUI

Assert Methods

`assert` (*boolean*)

`assert_nil` (*object*)

`assert_not_nil` (*object*)

uses
==

`assert_equal` (*expected, actual*)

`assert_not_equal` (*expected, actual*)

`assert_in_delta` (*expected-float, actual-float, delta*)

`assert_raise` (
 exception-list) { *block* }

`assert_nothing_raised` (
 [*exception-list*]) { *block* }

`assert_kind_of` (*class, object*)

`assert_instance_of` (*class, object*)

includes
superclasses

doesn't

`assert_respond_to` (*object, method-name-symbol*)

`assert_match` (*regexp, string*)

`assert_no_match` (*regexp, string*)

`assert_same` (*expected, actual*)

`assert_not_same` (*expected, actual*)

uses `equal?`
which tests
object identity,
not values

`flunk` () always fails

all of these methods take an optional last argument that is a custom message to be displayed if the test fails

Test::Unit Example (Cont'd)

- Unit test code – `sortertest.rb`

```
require 'sorter' # filename to be tested
require 'test/unit'
```

on next page

```
# This class provides unit tests for the Sorter module.
```

```
class SorterTest < Test::Unit::TestCase
```

```
  def test_sort_on_length
```

```
    my_array = ['red', 'green', 'blue']
```

```
    Sorter.sort_on_length(my_array)
```

```
    expected = ['red', 'blue', 'green']
```

```
    assert_equal(expected, my_array)
```

```
  end
```

```
  def test_something_bad
```

```
    flunk("something bad happened")
```

```
  end
```

```
end
```

assert_raise Example:

```
def test_divide_by_zero
  numerator, denominator = 5, 0
  assert_raise(ZeroDivisionError) {
    quotient = numerator / denominator
  }
end
```


Test::Unit Example

- Code to be tested – `sorter.rb`

```
# This module provides methods for sorting arrays.
module Sorter

  def self.sort_on_length(array)
    array.sort! { |a, b| a.length <=> b.length }
  end

end
```

Test::Unit Example (Cont'd)

- Commands to run tests

```
cd lib
testrb ../test/*.rb
```

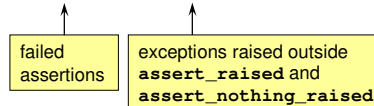
- `lib` directory contains code to be tested
- `test` directory contains unit test code

- Output

```
Loaded suite SorterTest.rb
Started
F.
Finished in 0.0 seconds.
```

```
1) Failure:
test_something_bad(SorterTest) [../test/SorterTest.rb:15]:
something bad happened.
```

```
2 tests, 2 assertions, 1 failures, 0 errors
```



Test Suites

- Test suites are sets of related tests
 - can have one test suite that runs all tests for the project
 - can have a hierarchy of test suites
- Test::Unit framework contains a **TestSuite** class
 - overkill?
- Simpler approaches
 - create a Ruby source file that contains a “require” for each test source file in the project
 - run that to run all the tests
 - use ***.rb** to specify tests to be run

Assertions Outside Unit Tests

- Assertions can be used outside unit tests to assert conditions that must hold at run-time
- If an assert fails, an **AssertionFailedError** is raised
- Example

```
require 'test/unit/assertions.rb'
include Test::Unit::Assertions

def quotient(numerator, denominator)
  assert(denominator != 0, 'denominator not zero')
  quotient = numerator / denominator
end

begin
  puts "quotient = #{quotient(19, 3)}"
  puts "quotient = #{quotient(19, 0)}"
rescue Test::Unit::AssertionFailedError
  puts $!
end
```

don't need begin/rescue,
but without it a backtrace
will be output

IDEs and Editors

- Popular IDEs used for Ruby development
 - ArachnoRuby
 - FreeRIDE
 - Komodo
 - Mondrian
 - RADRails
 - for Rails development
 - bundles Eclipse
 - Ruby Development Environment (RDE)
 - Windows only
 - RDT
 - Eclipse extension
 - TextMate
 - Mac only
- Popular editors used for Ruby development
 - emacs
 - jEdit
 - Notepad++
 - Scite
 - UltraEdit
 - vim

RDoc

- Tool that
 - extracts documentation from Ruby source files
 - outputs it several formats
- Can produce
 - HTML
 - hyperlinks display source code
 - YAML
 - used by ri command-line tool
 - ri is described in “Basic Tools” section
 - XML
 - could use XSLT to transform this to custom formats
 - CHM
 - Microsoft's HTML help format
- Can operate on classes/modules that have no comments
 - just extracts method names and parameters

Using RDoc

- Usage
 - **rdoc** [*options*] [*names*]
 - by default, generates HTML in **doc** directory for all source files in and below current directory
 - **names** can be names of source files or directories containing source files
 - if omitted, the current directory is used
 - for directories
 - all source files in and below it are processed unless a directory contains a **.document** file, in which case only files listed on separate lines in that file are processed
- HTML samples
 - browse <http://www.ruby-doc.org> and click [1.8.4 core](#)
 - also see sample screenshot ahead

RDoc Options

- h outputs help on the **rdoc** command
- f **format** specifies output format
 - choices are **chm**, **html**, **ri** and **xml**
 - defaults to **html** unless **-r** or **-R** option is used
- r generates **ri** compatible output under home directory
 - under UNIX, writes to **~** ←
 - under Windows XP, writes to **C:\Documents and Settings\username\.rdoc** ←
- R generates **ri** compatible output in a site-wide directory
 - writes to **\$RUBY_HOME/share/ri/1.8/site** ←
 - installers should copy **ri** documentation to here
- S affects how method source code is displayed in HTML
 - without this option, when a method name is clicked, its source is displayed in a browser popup window
 - with this option, when "source" link below a method name is clicked, its source is displayed inline on the HTML page
- o **dir** specifies an alternate output directory
- v outputs version number of **rdoc**

ri looks in these directories by default

RDoc Output

- When outputting ri output, RDoc writes
 - one folder for each class/module
 - contains a YAML file for the class/module and one YAML file for each method
 - `-i` at end of filename indicates an instance method
 - `-c` at end of filename indicates a class method
- For more information about other output formats

`rdoc --help-output`

- Try it!

```

cd Ruby/examples/TestUnit
HTML output {
  rdoc -S for HTML with "source" links
  browse Ruby/examples/testunit/doc/index.html
  click "Sorter" class
  click "Source" link below sort_on_length method
ri output {
  rdoc -r for ri output under home directory
  ri Sorter
  ri Sorter::sort_on_length

```

Sample Source File

```

require 'date'

# This class describes a person.
class Person
  attr_accessor :name, :birthdate

  def initialize(name, birthdate)
    @name = name
    @birthdate = birthdate
  end

  # Calculates the age of the person.
  def age
    Date.today.year - @birthdate.year
  end
end

```

Look for these comments, attributes and methods in the screenshot on the next page.

See pickaxe 202-207 for additional source file markup options such as specifying fonts, creating links and creating lists. See example of good documenting practice on pickaxe 213.

3

Sample RDoc HTML

RDoc Documentation - Mozilla Firefox

Files	Classes	Methods
person.rb	Person	age (Person) new (Person)

Class Person
In: person.rb
Parent: Object

This class describes a person.

Methods
age new

Attributes
birthdate [RW]
name [RW]

Public Class methods
→ new(name, birthdate)

Public Instance methods
→ age()
Calculates the age of the person.

Done

click these
to see source
in a pop-up
window

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

27

3

Logging

- Standard library contains **Logger** class
 - can write to files, **STDOUT** or **STDERR**
 - six predefined levels
 - DEBUG < INFO < WARN < ERROR < FATAL < UNKNOWN
 - supports automatic rolling of log files
 - can maximum logfile size before rotating and number of log files to keep
 - can specify frequency of rotation
 - 'daily', 'weekly' or 'monthly'
 - can control format of messages
- For more advanced logging, see **Log4r**
 - <http://log4r.rubyforge.org/>

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

28

Logging Example

```
require 'logger'

# Making this a global variable so other classes can share it.
$log = Logger.new('demo.log') ← can pass a String file name or an IO object
$log.level = Logger::DEBUG # lowest level that is logged default is DEBUG

class LoggerDemo
  def initialize
    $log.procname = 'LoggerDemo'
    $log.info('initialize called')
  end

  def do_it
    3.times do |i|
      $log.debug("in do_it, i = #{i}")
    end
  end
end

ld = LoggerDemo.new
ld.do_it
```

Output in demo.log:

```
I, [2005-08-28T19:09:08.190000 #3848] INFO -- LoggerDemo: initialize called
D, [2005-08-28T19:09:08.190000 #3848] DEBUG -- LoggerDemo: in do_it, i = 0
D, [2005-08-28T19:09:08.190000 #3848] DEBUG -- LoggerDemo: in do_it, i = 1
D, [2005-08-28T19:09:08.190000 #3848] DEBUG -- LoggerDemo: in do_it, i = 2
```

Debugging

- A line-oriented debugger is included with the standard Ruby installation
 - covered here
- Some IDEs provide “better” debuggers
- To start debugger
 - `ruby -rdebug [options] source-file arguments`
 - `-r` says to require the debug library
- Debugger commands
 - main commands are covered here
 - thread-related commands and a few others are not
 - for a full list, use help command (next page) or see pickaxe 173
 - no command to restart execution
 - must quit and restart debugger
 - can't save breakpoints and watches

also see Mr. Guid, a GUI front-end to this debugger, at <http://mr-guid.rubyforge.org/>

Some Debugger Commands

- **Help**

`h[elp]` – outputs a summary of debugger commands

- **Breakpoints and Watches**

`b[reak] [file:|class:]line`

– sets a breakpoint at a given line in a file or class (defaults to current file)

`b[reak] [file:|class:]method-name`

– sets a breakpoint at a given method in a file or class

`wat[ch] expr` – breaks when `expr` becomes true

`b[reak]` – lists current breakpoints and watchpoints

`del[ete] [number]` – deletes a given breakpoint/watchpoints or all of them

- **Execution**

`s[tep]` – steps in

`n[ext]` – steps over

`c[ont]` – continues

`fin[ish]` – steps out of current method

`q[uit]` or `exit` – exits debugger

pressing the **Enter** key **without** entering another debugger **command** **repeats** the last command; commonly used after **next** or **step**

Some Debugger Commands (Cont'd)

- **Context**

`l[ist] [start-end]`

– by default, lists 5 lines before, current line, and next 4 lines

`w[here]` or `f[rame]` – lists stack frames

`up` – moves up one stack frame

`down` – moves down one stack frame

- **Data**

`[p] expr` – evaluate `expr` in current context

need **p** when `expr` matches a debugger command

`v[ar] c[onst] name` – lists constants in a given class or module

`v[ar] g[lobal]` – lists global variables

`v[ar] i[nstance] obj` – lists instance variables in a given object

`v[ar] l[ocal]` – lists local variables

- **Methods** (useful for deciding where to break)

`m[ethod] i[nstance] obj` – lists instance methods of the given object

`m[ethod] name` – lists instance methods of the given class or module

Try It!

- **Debug fact.rb** ←

- cd to that directory
- enter the following commands

```

ruby -rdebug fact.rb 5
h (displays help on debugger commands)
l 1-20 (there are only 9 lines, so this lists them all)
b 3 (sets a breakpoint at the 3rd line)
c (continues execution until breakpoint is hit)
wat i < 3 (sets watch so execution will stop when i is less than 3)
c (continues execution until watch expression is true)
p i (prints the value of i which will be 2)
w (lists stack frames; line 9 called fact method; on line 5 within that frame)
up (moves up to the calling stack frame)
v l (lists local variables; there are none here)
down (moves down to stack frame of fact method)
v l (lists local variables; f, i and n)
s (steps to next line; n would have done the same since no method is called)
p i (prints value of i which will now be 1)
fin (exits fact method; would have stopped at next line in caller if there were one)
  
```

Linux Ruby installation includes this in the "sample" folder.
Windows One-Click Installer includes this in the "src/ruby-1.8.4/sample" folder.
A copy is in Ruby/Examples/debugger.

Profiling

- **A basic profiler is included with the standard Ruby installation**

- outputs to stdout
 - number of times each method is called
 - average time required for each run
 - total time required for all runs
- methods are sorted in descending order based on percentage of total time spent in them

- **To use profiler**

```
ruby -rprofile [options] source-file arguments
```

- **-r** says to require the profile library

- **Profiling adds overhead!**

- times reported will be longer than when running without profiler

- **Also consider Benchmark module**

- see pickaxe 170

3

Try It!

- Profile `fib.rb` ←

- cd to that directory
- enter the following command

```
ruby -rprofile fib.rb ← takes about 30 seconds to complete
```

- output will be similar to the following

6765 ← output from fib.rb (20th fibonacci number)

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
74.08	11.03	11.03	21891	0.50	9.20	Object#fib ←
10.81	12.64	1.61	21890	0.07	0.07	Fixnum#-
9.67	14.08	1.44	21891	0.07	0.07	Fixnum#<
5.44	14.89	0.81	10945	0.07	0.07	Fixnum#+
0.00	14.89	0.00	1	0.00	0.00	Fixnum#to_s
0.00	14.89	0.00	1	0.00	0.00	Module#method_added
0.00	14.89	0.00	2	0.00	0.00	IO#write
0.00	14.89	0.00	1	0.00	0.00	Kernel.print
0.00	14.89	0.00	1	0.00	14890.00	#toplevel
% of total time spent in this method	total seconds spent in this method and those above it	total seconds spent in this method	# of times this method was called	average ms per call excluding what it calls	average ms per call including what it calls	method name

the hot spot!
optimize this first

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

35

4

RubyGems Overview

- Preferred mechanism for packaging, distributing and installing Ruby libraries and applications
- A tool that
 - packages a Ruby library or application into single file called a “gem”
 - installs, updates and uninstalls gems
 - can update itself
- Gems are stored in and retrieved from local and remote repositories

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

36

Installing The RubyGems Tool

- Will eventually be a standard part of the Ruby install
- Windows One-Click Installer includes it
 - don't need to install it
- To install it ...
 - download from <http://rubygems.rubyforge.org>
 - unpack it
 - cd to the unpacked directory
 - run "**ruby setup.rb**"
 - must be root under Linux
 - if error message "No such file to load - ubygems (LoadError)", make sure RUBYOPT environment variable doesn't contain "rubygems"

RubyForge is the most popular place to store open source Ruby libraries and applications. You can store your own there so others can easily find and download them.

RubyGems Help

- Built-in help
 - run "**gem help**" for top-level help
 - run "**gem help commands**" for a list of supported commands
 - run "**gem help [command-name]**" for help on a particular command
 - run "**gem help examples**" for examples of usage
- See <http://docs.rubygems.org/> for
 - RubyGems User Guide
 - gem Command Reference
 - RubyGems Frequently Asked Questions
 - RubyGems GemSpec Reference
 - GemSpecs are files that provide information needed to create gems

Installing Gems

- To install a gem, run “**gem install gem-name**”
 - installs latest version unless **-v** is used to specify version
 - if gem being installed depends on others not installed yet, will prompt whether they should be installed also
 - unless **-y** option is used (see below)
- Options
 - d** generates RDoc documentation as part of install
 - l** installs from a local gem file, as opposed to a remote repository
 - only looks for gem in current directory
 - if neither **-l** (local) or **-r** (remote) is specified, will search both
 - source url** uses non-default URL for remote gems
 - default remote repository is `http://gems.rubyforge.org`
 - t** runs gem unit tests
 - if the gem has no tests, it outputs “There are no unit tests to run”
 - if any tests fail, it prompts whether to keep the gem
 - if no tests fail, there is no output and no indication that tests were run UNIX-like
 - y** installs gems on which this depends without prompting

Where Are Gems Installed?

- Gems are installed in
 - `$RUBY_HOME/lib/ruby/gems/1.8/gems/gem-name`**
 - where 1.8 is the version of Ruby used to install the gem
 - **RUBY_HOME**
 - under Linux, this is typically `/usr/local/lib/ruby` or `/opt/ruby`
 - under Windows, there isn't a standard location

Updating and Uninstalling Gems

- To update an already installed gem
 - run `gem update [gem-name]`
 - gets latest version of all installed gems or a named gem
 - supports `-d` (Rdoc) and `-t` (unit test) options like `gem install`
- To uninstall a gem
 - run `gem uninstall gem-name`
 - uninstalls all versions
 - include `-v version` to uninstall a specific version and leave others

Querying Gems

- To get a list of already installed gems
 - run `gem list`
- To get a list of available gems whose name contains a given string
 - run `gem search -r string`
 - for example, searching for `xml` finds the gem `xmlresume2x` and many others
- To get a list of gems on which a given gem depends
 - run `gem dependency gem-name`
 - for example, `activerecord` depends on `activesupport`
- To get a list of files associated with an installed gem
 - run `gem contents gem-name`
 - lists the full path to every file installed from the gem

ArachnoRuby has a GUI Gem browser

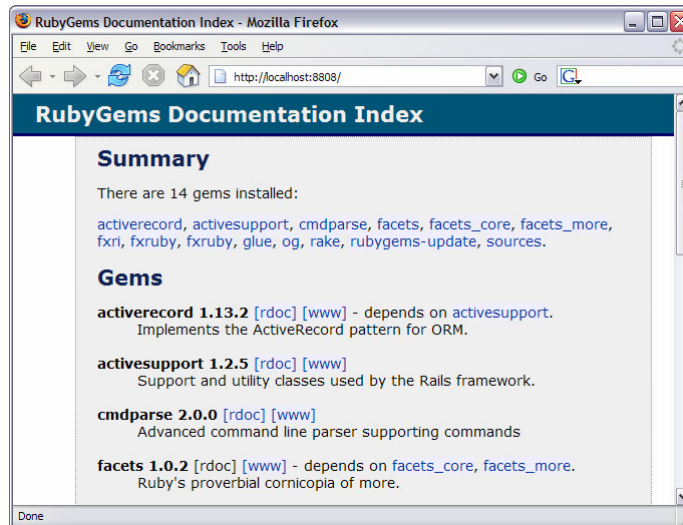
Querying and Updating the RubyGems Tool

- To get information about the RubyGem tool
 - run `gem env[environment]`
 - provides
 - version
 - local directory where gems are stored
 - can store multiple versions of the same gem
 - URL(s) used to retrieve remote gems
- To update the RubyGems tool
 - run `gem update --system`

Gem Documentation

- To generate RDoc HTML documentation for installed gems
 - for a specific gem, run `gem rdoc gem-name`
 - for all installed gems, run `gem rdoc --all`
 - may need to be root under Linux
- To view the documentation
 - start gem server by running `gem_server`
 - browse to `http://localhost:8808` to see a summary of all installed gems
 - see screenshot on next page
 - click `[rdoc]` links after name of each gem to view the RDoc files/classes/methods documentation
 - click `[www]` links after name of each gem to visit the associated web site

Documentation from Gem Server



Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

45

Using Gems In Applications

- Using latest version
 - approach #1: ask for newest version of a given gem


```
require_gem 'gem-name'
```

 - source file used is specified by the gemspec `autorequire` attribute
 - approach #2: ask for newest version of a given gem source file


```
require 'filename'
```
- Using a specific version


```
require_gem 'gem-name', 'version-spec'
```

 - ex. `require_gem 'runningcalc', '>= 1.2'`
- Specifying versions just once
 - many source files in an application often need same gem versions
 - solution is to specify all gem versions used by an application in one file, named `environment.rb` by convention
 - contents are same as previous example
 - add `require 'environment.rb'` in all source files that use the gems

see
p. 50

To use `require_gem`, need `"require 'rubygems'"` unless `RUBYOPT` env. variable contains `rubygems`. Windows One-Click Installer sets this; `ubygems.rb` is in `$RUBY_HOME/lib/ruby/site_ruby/1.8`

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

46

Version Numbers

- Conventions for gem versions
 - *major.minor.build*
 - start at 0.0.1
 - three kinds of changes
 - **implementation detail - build number increases**
 - implementation details change, but no API changes are made
 - **backwards-compatible change - minor number increases**
 - API additions are made, but the previous API still works
 - **incompatible change - major number increases**
 - API changes are made and something in the previous API no longer works
- Specifying required versions with `require_gem`
 - all normal relational operators are supported: `=, !=, <, <=, >, >=`
 - can specify more than one of these constraints
 - example: `'>= 4.2', '< 5.3'`
 - "pessimistic version constraint": `~>`
 - `'~> 2.3'` means latest version that is at least 2.3.0 and before 2.4.0

default when only a version number is specified

Creating Gems

- Place code in a standard directory structure
 - `lib` subdirectory for library/application source files
 - `test` subdirectory for unit test source files
 - `bin` subdirectory for shell scripts
 - `README` file that explains the purpose of the library/application and provides copyright and licensing information
 - `name.gemspec` file that provides information needed to create the gem, in either Ruby or YAML format
 - `Rakefile` file if Rake tool is used to build the library/application
 - a Ruby alternative to Make
- To create a new gem
 - run `gem build name.gemspec`

Gem Creation Example

- **runningcalc** directory contents

- README
- runningcalc.gemspec
- lib
 - calculator.rb
 - ChicagoFinish2004.gif
 - guiWithGrid.rb
- test
 - test_calculator.rb



Example GemSpec in Ruby Format

```
Gem::Specification.new do |s|
  s.name = 'runningcalc'
  s.summary = 'A Tk GUI for calculating information about running'
  s.version = '0.2.0'
  s.author = 'R. Mark Volkmann'
  s.email = 'mark@ociweb.com'
  s.has_rdoc = true
  s.extra_rdoc_files = ['README']

  # Include all files in the lib and test directories.
  s.files = Dir.glob('lib,test/**/*')
  # Exclude all CVS-related files, if any.
  s.files.reject! {|file| file.include? 'CVS'}

  # Specify unit test files.
  s.test_files = Dir.glob('test/*')

  # Specify source file to be loaded if require_gem is used.
  s.autorequire = 'calculator'

  # If this gem depended on another, this is how it could be specified.
  #s.add_dependency('other-gem-name', '>= version')
end
```

GemSpec Attributes

- **Commonly used attributes** (optional unless marked required)
 - `author` – name of the gem author
 - `dependencies` – name/version of others gems this one needs
 - `description` – detailed description (see summary)
 - `email` – email address of gem author
 - `files` – list of files that comprise the gem
 - `has_rdoc` – “true” if source files use RDoc; defaults to “false”
 - `homepage` – URL of project
 - `name` – name of gem; required
 - `required_ruby_version` – version of Ruby required to use the gem (ex. “>= 1.8.1”)
 - `rubyforge_project` – name of RubyForge project if any
 - `summary` – short description (see description); required
 - `test_files` – list of unit test files
 - `version` – version number of the gem; required
- **Complete list of attributes**
 - see <http://docs.rubygems.org/read/chapter/20>

Example Gem Usage

- **Using latest version available**

```
require_gem 'runningcalc' # using gem name
or
require 'calculator' # using source file name
```
- **Using an explicit version with `require_gem`**

```
require_gem 'runningcalc', '>= 0.2.0'
```
- **Exception raised if matching gem isn't found**
 - doesn't download and install it for you

Making Your Gems Available

- **Three ways**
 - make gem files available for download from a web page or FTP server
 - run your own `gem_server`
 - create a RubyForge project
 - the default remote `gem_server`
 - probably the best option

SWIG Overview

- **Generates wrapper code from C/C++ header files that allows C/C++ functions to be invoked from other languages**
 - **compiled languages**: C#, Java, Lua, Modula-3, Ocaml
 - **"scripting languages"**: Perl, PHP, Pike, Python, Ruby, TCL
 - **LISP**: Allegro CL, CLISP
 - **Scheme, a LISP dialect**
 - CHICKEN (compiles to C)
 - Guile (interpreter)
 - MzScheme (compiles to bytecode)
- **Motivations to use**
 - **saves time**
 - code generation avoids large amounts of tedious manual coding
 - **avoids errors**
 - code generation is less error prone than manual coding

SWIG Installation

- Download from <http://www.swig.org>
 - unzip and untar
- Windows installation
 - comes with a pre-built executable, so no installation is necessary
 - after unzipping, see [Doc/Manual/Windows.html](#) for more information
- Unix/Linux installation
 - some Linux distributions automatically install SWIG
 - if “which swig” doesn’t find it, follow these steps
 - cd to directory created from untar
 - `./configure`
 - `make`
 - `su root (or sudo su)`
 - `make install (requires root access)`

Example

- The following example shows how to use SWIG to
 - create a C++ object from Ruby
 - invoke methods on it from Ruby

5

Example C++ Header (Person.h)

```

#ifndef PERSON_H
#define PERSON_H

#include "boost/date_time/gregorian/gregorian.hpp"
#include <string>

class Person {

public:

    Person(const std::string& name);
    void setBirthday(const std::string& birthday);

    int getAge() const;
    boost::gregorian::date getBirthday() const;
    std::string getName() const;
    bool isOlderThan(const Person& person) const;

```

This example uses the Boost date_time library to store dates and perform calculations using them.

call this before calling
getAge, getBirthday
or isOlderThan

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

57

5

Example C++ Header (Cont'd)

```

private:

    boost::gregorian::date birthday_;
    std::string name_;

};

#endif

```

Copyright © 2006 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Tools

58

Example C++ Source (Person.cpp)

```
#include "Person.h"

using namespace boost::gregorian;
using namespace std;

Person::Person(const string& name) : name_(name) {
}

int Person::getAge() const {
    date today(day_clock::local_day());
    int days1 = today.day_of_year();
    int days2 = birthday_.day_of_year();

    int years = today.year() - birthday_.year();
    if (days1 < days2) --years;
    return years;
}
```

Example C++ Source (Cont'd)

```
date Person::getBirthday() const { return birthday_; }

string Person::getName() const { return name_; }

bool Person::isOlderThan(const Person& person) const {
    return birthday_ < person.birthday_;
}

void Person::setBirthday(const string& birthday) {
    birthday_ = from_string(birthday);
}
```

SWIG Input (Person.i)

SWIG directives start with "%"

```
%module RubyPerson
```

```
{
#include "Person.h"
}
```

everything between %{ and %} is copied into the generated C++ wrapper code (Person_wrap.cxx in this case)

```
{
#include "std_string.i"
#include "Person.h"
}
```

needed to pass STL string objects as parameters

SWIG operates on these function and class declarations

can put C or C++ style comments in these files

There are several SWIG directives that are not used in this simple example.
For code insertion,
%{ ... }, %inline, %init, %wrapper
For documentation,
%title, %section, %subsection, %subsubsection

Running SWIG For Ruby Under Linux

Generate C++ wrapper code.

```
swig -c++ -ruby *.i processes Person.i
```

Compile wrapper code.

```
rubydir=/usr/local/lib/ruby/1.8/i686-linux
```

```
gcc -c -fpic *.cpp *_wrap.cxx \
-I. -I$rubydir -I/usr/local/include/boost_1_33_0
```

Create shared library containing wrapper code.

```
gcc -shared -fpic Person.o *_wrap.o \
-lboost_date_time-gcc-1_33 -lstdc++ \
-o RubyPerson.so
```

Example Ruby Source (main.rb)

```
require 'RubyPerson.so'
include RubyPerson ← RubyPerson is a Ruby module generated by SWIG

p1 = Person.new('Mark Volkmann')
p1.setBirthday('1961/4/16')
puts "#{p1.getName()} is #{p1.getAge()} years old."

p2 = Person.new('Amanda Volkmann')
p2.setBirthday('1985/7/22')
puts "#{p2.getName()} is #{p2.getAge()} years old."

puts "#{p1.getName()} is older than #{p2.getName()}? " +
      "#{p1.isOlderThan(p2)}"
```

Running Ruby

```
ruby main.rb
```

Output:

```
Mark Volkmann is 44 years old.
Amanda Volkmann is 20 years old.
Mark Volkmann is older than Amanda Volkmann? true
```


Thank You For Attending!

- For more information and to report errors in slides
 - [contact me at mark@ociweb.com](mailto:mark@ociweb.com)
- Other learning opportunities in St. Louis
 - **St. Louis Ruby User Group**
 - meets on the 4th Tuesday of each month at 6PM at OCI
 - <http://www.stlruby.org/>
 - updates to these slides will be here!
 - **OCI training**
 - <http://www.ociweb.com/education>
 - “Ruby Programming”
 - <http://www.ociweb.com/education/services/descrip/ESOS07-01.html>
 - first offering is an evening course for four nights that starts on 4/11/06
 - “Ruby on Rails Web Development”
 - coming soon