Ruby, an Introduction

Jim Menard, jimm@io.com

July 10, 2001

Contents

1 Introductions

1.1 Me

"But enough about me . . . what do you *think about me?"*

1.2 Ruby

A pure-OO scripting language

1.2.1 History and Pedigree

- Developed by Yukihiro "Matz" Matsumoto
- Looking for OO scripting language; goal to be better than Perl
- Perl too messy (a "Swiss Army chain saw"), Python not OO enough
- Take Smalltalk, make file based w/familiar syntax, add best features of many other languages (regular expressions, iterators), abstract many features into classes (`Regexp`) and mixin modules (`Enumerable`)
- Released 1993, hit USA 2000
- More popular than Python in Japan

2 Why Ruby?

- Simple: easy to learn and maintain
- Powerful
- Language stays out of your way
- Rich libraries
- Rapid development
- Helpful community (1-hour turnaround from matz)
- Open Source
- Fun

2.1 Why Not?

- Performance (though rivals Perl and Python; can wrap or be wrapped by C code)
- Lack of frameworks (e-commerce; only one which is immature)
- Current internal standards
- Cross-platform GUIs (Tk, Qt, Gnome) not widely accepted
- Threading model

3 Language Features

There are too many to list. I'll try anyway.

- Simple, consistent syntax; Principle of Least Surprise (POLS): things work the way you expect them to, with few special cases or exceptions
- Dynamically typed/late binding (don't need to declare variable types; can change your design without changing types everywhere; decisions deferred until run time)
- Everything an object or a method (even operators are methods)
- Single inheritance with mixin modules (name space, method implementation) and object-specific methods
- Blocks and closures

```
1 array = [1, 2, 3]
2 x = 5
3 array.collect! { | elem | elem + x }
4 p array                          # => [6, 7, 8]
```

- Enumeration extracted into a mixin module available to all classes and independent of the language syntax (`each`, `each_with_index`, `sort`, `collect`, `detect`, `reject`, `select`, `entries`, `find`, `grep`, `include?`, `map`, `max`, `reject`, more)

```
1 array.each { | elem | puts elem }
2 hash.each { | key, val | puts "#{key} => #{val}" }
3 file.each { | line | puts line }
4 array.each_with_index { | elem, i |
5     puts "array[#{i}] = #{elem}"
6 }
7 array.include?(42)
```

- Ranges

```
1 (0 .. 9).each { | i | sum += i }
2 ('a' .. 'z').each { | str | puts str }
3 (0 ... array.size).each { | i | puts array[i] }
```

- Metaclasses
- Introspection (reflection)
- Method aliasing
- Garbage collection
- Exception handling
- C integration
- Simple naming conventions: `@instanceVar`, `@@classVar`, `$global`, `CONSTANT`, `everythingElse`
- Loadable extension libraries (if OS allows)
- OS-independent threading
- Portable
  - Linux, most *NIX
  - Macintosh (OS 9 and OS X)
  - BeOS
  - OS/2
  - DOS, Windows 95/98/NT/2K
    - Install Shield
    - console and non-console

    * relies on Cygwin libs (bundled with Install Shield distribution)
    * `Win32OLE` (Windows Automation, properties, named arguments)

```
1 ie = WIN32OLE.new('InternetExplorer.Application')
2 ie.visible = true
3 ie.gohome
```

    * Can access any function in any DLL
    * Ask me for code to access an ODBC database and a client/server for remote ODBC access from any box running Ruby

- Libraries (see Tools below)
- Security (levels of safety using `$SAFE`; `taint/untaint/freeze`)
- Internationalization (I18N) and multilingualization (M17N); `String` class supports UTF and Kanji; whole system will in next major release

4 Comparison With Other Languages

Much of this section is stolen from the Ruby site (see References below).

4.1 Java

- Ruby is interpreted
- Everything is an object (no int/Integer distinction)
- Modules used to share implementations between classes
- Variables and expressions not statically typed
- Ruby's mixin modules similar to Java interfaces, but provide method implementation
- Less verbose; for example:
  - Reading lines from a file

```
1 // ================
2 // Java
3
```

```
 4 import java.io.*;
 5
 6 try {
 7     BufferedReader in =
 8         new BufferedReader(new FileReader(fileName));
 9     String line;
10     while ((line = in.readLine()) != null) {
11         // Use the line...
12     }
13 }
14 catch (FileNotFoundException fnfe) {
15     System.err.println(fnfe.toString());
16 }
17 catch (IOException ioe) {
18     System.err.println(ioe.toString());
19 }
20
21 # ================
22 # Ruby
23
24 begin
25     IO.foreach(fileName) { | line |
26         # Use the line...
27     }
28 rescue
29     $stderr.puts $!
30 end
```

4.2 Perl

- Ruby is easier to learn, use, and maintain
- No $@% code (in Ruby, '`@`' and '`$`' denote variable scope, not type)
- Ruby's default scoping rules obviate need for '`my`'
- Ruby copies many good things from Perl: extended regular expressions, `$_` shortcuts, and more

4.3 Smalltalk

- Ruby is file based

- Ruby syntax more familiar
- Only primitive IDEs so far

4.4 Python

- Ruby uses conservative statement structure, using '`end`'
- '`self.`' not necessary; Ruby's instance variables prefixed by '`@`'
- Python separates types and classes; Ruby treats them the same
- Python types are more limited
  - No inheritance/subclassing
  - Cannot add methods to existing types
- Ruby has a better (or "real") closure feature.
- Object attributes are not accessible by default in Ruby.
- Ruby converts small integers and long integers automatically.
- Ruby does not have tuples
- Ruby has more natural operator overloading
- Ruby uses mark-and-sweep GC, unlike Python's ref-counting GC (this means no INCREF, DECREF required for extensions)
- Extensions for Ruby written in C/C++ can easily define Ruby classes
- Ruby has a loop abstraction using block; e.g.

```
1 10.times do
2     # ...
3 end
```

- You can define your own arbitrary iterator.
- Ruby's block for method can be used more than iterator; e.g.

```
1 mutex.synchronize do
2      # .. critical process ..
3 end
```

- Ruby provides method combination using '`super`'.
- Ruby is often faster than Python.

5 Code Examples

Here we have four code examples. The first three are relatively simple "textbook" examples that perform the same task in Java, C++, Smalltalk, Perl, and Ruby. The last is a slightly more complex example in Ruby only.

The Smalltalk code below was written using Squeak, a free implementation of Smalltalk. Sorry, but there are no Python examples. I don't know that language well enough. *ObPython: "My hovercraft is full of eels."*

5.1 A class with a few simple methods

The first example defines the same class in five different languages. In each, we define the class, create an instance, and print the string representation of the instance.

The example class is a `Song`. It has has a constructor, two instance variables, accessor methods ("getters" and "setters"), and a method for representing instances as strings.

```
 1 // ================
 2 // Java
 3
 4 public class Song {
 5
 6     protected String name;
 7     protected int lengthInSeconds;
 8
 9     Song(String name, int len) {
10         this.name = name;
11         lengthInSeconds = len;
12     }
13
14     public String getName() { return name; }
15     public void setName(String str) { name = str; }
16
17     public int getLengthInSeconds()
18         { return lengthInSeconds; }
19     public void setLengthInSeconds(int secs)
20         { lengthInSeconds = secs; }
21
22     public String toString() {
```

```
23          return name + " (" + lengthInSeconds + " seconds)"
24      }
25
26      // Create and print
27      public void main(String[] args) {
28          s = new Song("name", 60);
29          System.out.println(s);
30      }
31  }
32
33  // ================
34  // C++
35
36  // Song.h
37  #ifndef Song_h
38  #define Song_h
39
40  #include <iostream>
41  #include <string>
42
43  class Song {
44  public:
45      Song(const char * nameStr = 0, int len = 0)
46          : name(nameStr ? nameStr : ""),
47            lengthInSeconds(len) {}
48
49      // The default copy constructor, destructor, and
50      // operator= are all acceptable. I'm glad, 'cause
51      // it's a pain to have to write them.
52
53      const char * getName() const { return name.data(); }
54      void setName(const char *str) { name = str; }
55
56      int getLengthInSeconds() const
57          { return lengthInSeconds; }
58      void setLengthInSeconds(int len)
59          { lengthInSeconds = len; }
60
61  protected:
62      string name;
```

```
63      int lengthInSeconds;
64
65      friend ostream &
66          operator<<(ostream &out, const Song &s);
67  };
68
69  ostream & operator<<(ostream &out, const Song &s) {
70      return out << s.name << " ("
71                 << s.lengthInSeconds << " seconds)";
72  }
73
74  #endif Song_h
75
76  // Song.cpp
77  #include "Song.h"
78
79  main()
80  {
81      Song s("name", 60);
82      cout << s << endl;
83  }
84
85  " ================"
86  "Smalltalk"
87
88  Object subclassNamed: #Song
89      instanceVariables: 'name lengthInSeconds'
90      classVariables: ''
91  "..."
92
93  name
94      ^ name
95
96  name: aString
97      name := aString
98
99  lengthInSeconds
100     ^ lengthInSeconds
101
102 lengthInSeconds: aNumber
```

```
103     lengthInSeconds := aNumber
104
105 printOn: aStream
106     aStream nextPutAll: name.
107     aStream nextPut: $(.
108     aStream nextPutAll: lengthInSeconds printString.
109     aStream nextPutAll: ') seconds'
110
111 "Create and print a song.
112 Copy this code into a workspace and execute it."
113 s := Song new name: 'name'; lengthInSeconds: 60.
114 Transcript show: s asString; cr.
115
116 # ================
117 # Perl
118
119 package Song;
120
121 sub new {
122     my($class, $name, $len) = @_;
123     my $self = {};
124     $self->{'name'} = $name;
125     $self->{'lengthInSeconds'} = $len;
126     bless $self, class;
127     return $self;
128 }
129
130 sub name {
131     my($self) = shift;
132     if (@_) { $self->{'name'} = shift }
133     return $self->{'name'};
134 }
135
136 sub lengthInSeconds {
137     my($self) = shift;
138     if (@_) { $self->{'lengthInSeconds'} = shift }
139     return $self->{'lengthInSeconds'};
140 }
141
142 sub toString {
```

```
143     my($self) = shift;
144     return $self->name() . "(" . $self->lengthInSeconds()
145         . ") seconds";
146 }
147
148 # Create and print
149 $s = Song->new('name', 60);
150 print $s->toString() . "\n";
151
152 # ================
153 # Ruby
154
155 class Song
156
157     # Not only declare instance variables (which is
158     # unnecessary) but also create accessor methods
159     # (getters and setters). "attr_reader" creates
160     # just getters and "attr_writer" creates just
161     # setters.
162     attr_accessor :name, :lengthInSeconds
163
164     # The constructor, sort of. This method is called
165     # by the class method "new".
166     def initialize(name, len)
167         @name = name
168         @lengthInSeconds = len
169     end
170
171     def to_s
172         return "#{name} (#{lengthInSeconds} seconds)"
173     end
174 end
175
176 # Create and print. Only run this code if this file
177 # is being executed directly, else ignore it.
178 if $0 == __FILE__
179     s = Song.new('name', 60)
180     puts s
181 end
```

5.2 Iterating over a collection

In this example we will create a linear collection of `Song` instances, add a song, and iterate over the collection.

The Java version uses the J2EE collection classes. The C++ version uses the Standard Template Library (STL) for its vector and iterator classes.

```
 1 // ================
 2 // Java
 3
 4 import java.util.*;
 5
 6 ArrayList songs = new ArrayList();
 7 songs.add(new Song("name", 60));
 8
 9 for (Iterator iter = songs.iterator(); iter.hasNext(); ) {
10     Song s = (Song)iter.next();        // Yuck!
11     // Do something with s...
12 }
13
14 // ================
15 // C++
16
17 #include <vector>
18 #include "Song.h"
19
20 main()
21 {
22     vector<Song> songs;
23     songs.push_back(Song("name", 60));
24
25     vector<Song>::iterator iter(songs.begin());
26     for (; iter != songs.end(); ++iter) {
27         Song s = *iter;
28         // Do something with s...
29     }
30 }
31
32 " ================"
33 " Smalltalk"
```

```
34
35 songs := OrderedCollection new.
36 songs add: (Song new name: 'name'; lengthInSeconds: 60).
37 songs do: [ :s |
38     "Do something with s..."
39 ].
40
41 # ================
42 # Perl
43
44 @songs = ();
45 push(@songs, Song->new("name", 60));
46 foreach $s (@songs) {
47     # Do something with $s...
48 }
49
50 # ================
51 # Ruby
52
53 songs = []
54 songs << Song.new("name", 60)
55 songs.each { | s |
56     # Do something with s...
57 }
```

5.3 Reading a file

In this example, we will manipulate a comma-delimited data file by opening it, reading each line, separating each line into columns, and printing the columns as a SQL INSERT statement.

For simplicity's sake, we will assume that the data does not contain any comma characters. This code does handle "empty" columns (two consecutive commas). Because of this, we often can't use the most obvious or clean solution (for example, a `StringTokenizer` in Java or `strtok()` in C++).

I have Ruby, Perl, and Smalltalk classes that handle Excel comma- and tab-delimited data—quotes and all—that are yours for the asking. A mini state machine is necessary to handle the quotes and delimiter chars properly.

```
1 // ================
2 // Java
```

```
 3
 4 import java.io.*;
 5 import java.util.*;
 6
 7 public class DataFileReader {
 8
 9 public static void main(String[] args) {
10     DataFileReader dfr = new DataFileReader();
11     dfr.readFile(args[0]);
12 }
13
14 public void readFile(String fileName) {
15     try {
16         BufferedReader in =
17             new BufferedReader(new FileReader(fileName));
18         String line;
19         while ((line = in.readLine()) != null) {
20             Collection list = splitIntoColumns(line);
21             printAsInsertStatements(list);
22         }
23     }
24     catch (FileNotFoundException fnfe) {
25         System.err.println(fnfe.toString());
26     }
27     catch (IOException ioe) {
28         System.err.println(ioe.toString());
29     }
30 }
31
32 public Collection splitIntoColumns(String line) {
33     ArrayList array = new ArrayList();
34
35     // Using StringTokenizer here is almost useless. It
36     // either skips or returns multiple adjacent commas,
37     // but doesn't report the emptiness between as empty
38     // data columns.
39     int pos = line.indexOf(",");
40     while (pos != -1) {
41         array.add(line.substring(0, pos));
42         line = line.substring(pos + 1);
```

```
43          pos = line.indexOf(",");
44      }
45      if (line.length() > 0)
46          array.add(line);
47
48      return array;
49  }
50
51  public void printAsInsertStatements(Collection list) {
52      System.out.print("insert into table values (\n\t");
53      boolean first = true;
54
55      Iterator iter = list.iterator();
56      while (iter.hasNext()) {
57          String col = (String)iter.next();
58
59          if (first) first = false;
60          else System.out.print(",\n\t");
61
62          System.out.print("'" + col + "'");
63      }
64      System.out.println("\n)");
65  }
66
67  }
68
69  // ================
70  // C++
71
72  #include <string>
73  #include <iostream>
74  #include <fstream>
75  #include <vector>
76
77  static const int BUFSIZ = 1024;
78
79  vector<string> splitIntoColumns(char *line);
80  void printAsInsertStatements(vector<string> &list);
81
82  void
```

```
 83 main(int argc, char *argv[])
 84 {
 85     ifstream in(argv[1]);
 86     char line[BUFSIZ];
 87     while (!in.eof()) {
 88         in.getline(line, BUFSIZ);
 89         if (strlen(line) == 0)
 90             break;
 91         vector<string> list = splitIntoColumns(line);
 92         printAsInsertStatements(list);
 93     }
 94 }
 95
 96 vector<string>
 97 splitIntoColumns(char *line)
 98 {
 99     vector<string> list;
100
101     // Using strtok() here is useless. It skips multiple
102     // adjacent commas and doesn't report the emptyness
103     // between as empty data columns.
104     char *nextComma = index(line, ',');
105     while (nextComma != 0) {
106         list.push_back(string(line, nextComma - line));
107         line = nextComma + 1;
108         nextComma = index(line, ',');
109     }
110     if (strlen(line) > 0)
111         list.push_back(string(line));
112
113     return list;
114 }
115
116 void
117 printAsInsertStatements(vector<string> &list)
118 {
119     cout << "insert into table values (" << endl << "\t";
120     bool first = true;
121
122     vector<string>::iterator iter(list.begin());
```

```
123     for ( ; iter != list.end(); ++iter) {
124         string col = *iter;
125 
126         if (first) first = false;
127         else cout << "," << endl << "\t";
128 
129         cout << "'" << col << "'";
130     }
131     cout << endl << ")" << endl;
132 }
133 
134 
135 " ================
136 " Smalltalk"
137 
138 Object subclass: #DataFileReader
139     instanceVariableNames: ''
140     classVariableNames: ''
141     poolDictionaries: ''
142     category: 'Jim-Utils'
143 
144 readFile: fileName
145     | inStream outStream list |
146     inStream :=
147         CrLfFileStream readOnlyFileNamed: fileName.
148     outStream :=
149         FileStream newFileNamed: fileName, '.out'.
150     [inStream atEnd] whileFalse: [
151         list := self parseNextLine: inStream.
152         self output: list onStream: outStream.
153     ].
154 
155 parseNextLine: aStream
156     "Read columns in line and shove them into a list.
157      Return the list."
158     | list line lineStream |
159     list := OrderedCollection new. "Create empty list."
160     line := aStream nextLine. "Read line from input file."
161 
162     "Read columns in line and shove into list."
```

```
163     lineStream := ReadStream on: line
164                             from: 1
165                               to: line size.
166     [lineStream atEnd] whileFalse: [
167         list add: (lineStream upTo: $,).
168     ].
169     ^ list
170
171 output: list onStream: outStream
172     "Output insert statement."
173     outStream nextPutAll: 'insert into table values (
174     '''.
175     list do: [ :col |
176         outStream nextPutAll: col
177     ] separatedBy: [
178         outStream nextPutAll: ''',
179     '''.
180     ].
181     outStream nextPutAll: '''
182 )
183 '.
184
185 "Finally, create and use one of these suckers. Type
186  this in a workspace, select it, and choose 'do it'.
187  Actually, this code can be anywhere, including in
188  a comment. Just select it and execute it."
189 DataFileReader new readFile: 'foo.dat'.
190
191 # ================
192 # Perl
193
194 #! /usr/bin/perl
195
196 open(FILE, $ARGV[0]) ;
197 while (($line = <FILE>) ne '') {
198     chomp($line);
199     print "insert into table values (\n\t'";
200     print join("',\n\t'", split(/,/, $line));
201     print "',\n)\n";
202 }
```

```
203 close(FILE);
204
205 # ================
206 # Ruby
207
208 #! /usr/local/bin/ruby
209
210 # File is a subclass of IO. "foreach" is a class method
211 # that opens the file, executes the block of code once
212 # for each line, and closes the file.
213 IO.foreach(ARGV[0]) { | line |
214     line.chomp!()
215     puts "insert into table values ("
216     print "\t'"
217     print line.split(/,/).join("',\n\t'")
218     puts "'\n)"
219 }
```

5.4 Parsing XML

This final example is Ruby-only. It shows a "real" script performing a slightly more complex task: parsing XML and capturing and printing any errors returned by the parser. The XML isn't processed by this script.

This is one of the example scripts that comes with NQXML.

```
 1 #! /usr/bin/env ruby
 2 #
 3 # usage: parseTestStream.rb file_or_directory
 4 #
 5 # This script runs the streaming parser over the specified
 6 # file or all .xml files within the specified directory.
 7 #
 8 # If an NQXML::ParserError is seen, an error message is
 9 # printed and parsing stops.
10 #
11
12 # Start looking for NQXML classes in the directory above
13 # this one. This forces us to use the local copy of NQXML,
14 # even if there is a previously installed version out
15 # there somewhere. (Since this script comes as an example
```

```
16 # with NQXML, we want to make sure we are using the latest
17 # version that's right here, not one previously installed
18 # elsewhere.)
19 $LOAD_PATH[0, 0] = '..'
20
21 require 'nqxml/streamingparser'
22
23 def testParser(file)
24     print "Parsing file #{file}..."
25     $stdout.flush()
26     file = File.open(file, 'r')
27     parser = NQXML::StreamingParser.new(file)
28     begin
29         # Just let parser run through the XML
30         parser.each { | entity | }
31     rescue NQXML::ParserError => ex
32           puts "\n  NQXML parser error, line #{ex.line}" +
33             " column #{ex.column}: #{$!}"
34         return
35     rescue
36         puts "\n  Non-parser error: #{$!}"
37         return
38     ensure
39         # An "ensure" is like Java's "finally" clause:
40         # No matter what happens, this code gets executed.
41         # Instead of using an "ensure" we could have used
42         # a block as an argument to File.open. At the end
43         # of the block the file would have been closed
44         # automagically.
45         file.close() unless file.nil?
46     end
47     puts 'OK'
48 end
49
50 # Start of main code
51
52 # If we have a command-line argument, grab it and
53 # remove the trailing slash (if any). If no argument
54 # is specified, use '.' (the current directory).
55 DIR = ARGV[0] ? ARGV[0].gsub(/\/$/, '') : '.'
```

```
56
57 if File.directory?(DIR)
58     Dir.entries(DIR).each { | f |
59         testParser("#{DIR}/#{f}") if f =~ /\.xml$/
60     }
61 else
62     testParser(DIR)
63 end
```

6 Practical Ruby Experience

- Two minutes to light bulb, two days to fanatical zealot
- Data massagers (from Perl to Ruby)
- Job Editor (Apache, mod_ruby, eRuby)
- NQXML (RubyUnit)
- XMLRPC (1/2 day to implement)
- Mecha
- RuBoids (KDE, OpenGL)
- ICE test syndicator/subscriber (coding from specs)

7 Tools

- Socket and networking classes (HTTP, FTP, SMTP, POP)
- XML (NQXML (pure Ruby), XMLParser (wrapper around James Clark's "expat" library))
- Web (mod_ruby & eRuby)
- SOAP and XMLRPC
- `Singleton`, `Delegator` and more support common OO design patterns
- `Mutex` and `ConditionVariable` (in addition to `Thread::critical=`)
- RubyUnit (supports XP and test-first methodologies)

- Win32
- Databases (Oracle, PostgreSQL, MySQL, MS SQL Server through Win32)
- Frameworks (IOWA, Facet)
- GUI (Tk, KDE, Gnome, OpenGL, curses)
- Distributed objects (around 200 lines of Ruby), Rinda
- irb
- Debugger
- RDTool (like Perl's POD)
- SWIG

8 Advantages and Disadvantages

8.1 Advantages

- Powerful language with simple, consistent syntax
- Easy to learn, especially if you already know Perl, Python, or Smalltalk (only slightly less so if you know Java, C++, or Objective-C)
- Mature (seven years old)
- Rich libraries
- Helpful community
- Easy to install

8.2 Disadvantages

- Not well known; not many installations (but remember when you had to fight to use Perl, Java, or C++?)
- RAA not yet as big or automated as CPAN
- Docs still sparse (or in Japanese)
- One book (more on the way)

- Threads implementation not native
- Few experienced coders

9 Resources

- This talk, as PDF and HTML:
  `http://www.io.com/~`jimm/downloads/rubytalk/
- Ruby home page: `http://www.ruby-lang.org/en/`
- Ruby Application Archive (RAA):
  `http://www.ruby-lang.org/en/raa.html`
- "Programming Ruby" by David Thomas & Andrew Hunt, Addison Wesley Longman (this book is online and is published under the Open Publication license):
  `http://www.pragmaticprogrammer.com/ruby/`
- *Ruby: a new language*
  Introducing the latest open source gem from Japan:
  `http://www-106.ibm.com/developerworks/library/ruby.html`
- Ruby Central: `http://www.rubycentral.com/`
- Ruby Cookbook: `http://www.rubycookbook.org/`
- Ruby Garden: `http://www.rubygarden.org/`
- `comp.lang.ruby`
- Mailing lists
- Squeak (free Smalltalk): `http://www.squeak.org/`
- NQXML: `http://www.io.com/~jimm/downloads/nqxml/`
- Me: Jim Menard, `jimm@io.com`, `http://www.io.com/~jimm/`