# Chapter 28

# Standard Library

The Ruby interpreter comes with a large number of classes, modules, and methods built in—they are available as part of the running program. When you need a facility that isn't part of the built-in repertoire, you'll often find it in a library that you can `require` into your program. Sometimes you'll need to download one of these libraries (perhaps as a Ruby gem).

However, Ruby also ships as standard with a large number of libraries. Some of these are written in pure Ruby and will be available on all Ruby platforms. Others are Ruby extensions, and some of these will be present only if your system supports the resources that they need. All can be included into your Ruby program using `require`. And, unlike libraries you may find on the Internet, you can pretty much guarantee that all Ruby users will have these libraries already installed on their machines.

Ruby 1.9 has more than 100 standard libraries included in the distribution. For each of these libraries, this section shows a one- or a two-page summary. For each library, we give some introductory notes and typically give an example or two of use. You won't find detailed method descriptions here; for that, consult the library's own documentation.

It's all very well suggesting that you "consult the library's own documentation," but where can you find it? The answer is that it depends. Some libraries have already been documented using RDoc (see Chapter 19). That means you can use the ri command to get their documentation. For example, from a command line, you may be able to see the following documentation on the escapeHTML method in the CGI standard library member:

```
% ri CGI.escapeHTML
------------------------------------------------ CGI::escapeHTML
     CGI::escapeHTML(string)
------------------------------------------------------------------
     Escape special characters in HTML, namely &¨<>

       CGI::escapeHTML('Usage: foo "bar" <baz>')
          # => "Usage: foo &quot;bar&quot; &lt;baz&gt;"
```

If there's no RDoc documentation available, the next place to look is the library itself. If you have a source distribution of Ruby, these are in the `ext/` and `lib/` subdirectories. If instead you have a binary-only installation, you can still find the source of pure-Ruby

library modules (normally in the `lib/ruby/1.9/` directory under your Ruby installation). Often, library source directories contain documentation that the author has not yet converted to RDoc format.

If you still can't find documentation, turn to Google. Many of the Ruby standard libraries are also hosted as external projects. The authors develop them stand-alone and then periodically integrate the code into the standard Ruby distribution. For example, if you want detailed information on the API for the YAML library, googling *yaml ruby* may lead you to `http://yaml4r.sourceforge.net`. After admiring *why the lucky stiff's* artwork, a click will take you to his 40+ page reference manual.

The next port of call is the `ruby-talk` mailing list. Ask a (polite) question there, and chances are that you'll get a knowledgeable response within hours. See page 894 for pointers on how to subscribe.

And if you *still* can't find documentation, you can always follow Obi Wan's advice and do what we did when documenting Ruby—use the source. You'd be surprised at how easy it is to read the actual source of Ruby libraries and work out the details of usage.

## 1.9 Library Changes in Ruby 1.9

These are the library changes in Ruby 1.9:

- Much of the Complex and Rational libraries are now built in to the interpreter. However, requiring the external libraries adds some functionally. In the case of Rational, this functionality is minimal.
- The CMath library has been added.
- The Enumerator library is now built in.
- The Fiber library has been added (it adds coroutine support to fibers).
- ftools have been removed (and replaced by fileutils).
- The Generator library has been removed (use fibers).
- Notes on using irb from inside applications have been added.
- jcode has been removed in favor of built-in encoding support.
- The json library is added.
- The matrix library no longer requires that you include mathn.
- The mutex library is now built in.
- parsedate has been removed. The Date class handles most of its functionality.
- readbytes has been removed. IO now supports the method directly.
- A description of Ripper has been added.
- A description of SecureRandom has been added.

- I've omitted the shell library, because it seems more like a curiosity than something folks would use (and it's broken under 1.9).
- The soap library has been removed.
- I've omitted the sync library. It is broken under 1.9, and the monitor library seems to be cleaner.
- Win32API is now deprecated in favor of using the DL library.

Library

## Abbrev — Generate Sets of Unique Abbreviations

Given a set of strings, calculates the set of unambiguous abbreviations for those strings and returns a hash where the keys are all the possible abbreviations and the values are the full strings. Thus, given input of “car” and “cone,” the keys pointing to “car” would be “ca” and “car,” and those pointing to “cone” would be “co,” “con,” and “cone.”

An optional pattern or a string may be specified—only those input strings matching the pattern, or beginning with the string, are considered for inclusion in the output hash.

Including the Abbrev library also adds an abbrev method to class Array.

- Shows the abbreviation set of some words:

Download **samples/slabbrev_1.rb**

```
require 'abbrev'

Abbrev::abbrev(['ruby', 'rules'])   # =>   {"rub"=>"ruby",
                                           "rule"=>"rules",
                                           "rul"=>"rules",
                                           "ruby"=>"ruby",
                                           "rules"=>"rules"}

%w{ car cone }.abbrev               # =>   {"ca"=>"car",
                                           "con"=>"cone",
                                           "co"=>"cone", "car"=>"car",
                                           "cone"=>"cone"}

%w{ car cone }.abbrev("ca")         # =>   {"ca"=>"car", "car"=>"car"}
```

- A trivial command loop using abbreviations:

Download **samples/slabbrev_2.rb**

```
require 'abbrev'
COMMANDS = %w{ sample send start status stop }.abbrev
while line = gets
  line = line.chomp
  case COMMANDS[line]
  when "sample":  # ...
  when "send":    # ...
  # ...
  else
    STDERR.puts "Unknown command: #{line}"
  end
end
```

Library

## Base64 — Base64 Conversion Functions



Performs encoding and decoding of binary data using a Base64 representation. This allows you to represent any binary data in purely printable characters. The encoding is specified in RFC 2045 (http://www.faqs.org/rfcs/rfc2045.html) and RFC 4648 (http://www.faqs.org/rfcs/rfc464

- Encodes and decodes strings. Note the newlines inserted into the Base64 string.

  Download **samples/slbase64_1.rb**

  ```
  require 'base64'
  str = "Now is the time for all good coders\nto learn Ruby"
  converted = Base64.encode64(str)
  puts converted
  puts Base64.decode64(converted)
  ```

  *produces:*

  ```
  Tm93IGlzIHRoZSBOaW1lIGZvciBhbGwgZ29vZCBjb2RlcnMKdG8gbGVhcm4g
  UnVieQ==
  Now is the time for all good coders
  to learn Ruby
  ```

- Now uses RFC 4648 variants:

  Download **samples/slbase64_2.rb**

  ```
  require 'base64'
  str = "Now is the time for all good coders\nto learn Ruby"
  converted = Base64.strict_encode64(str)
  puts converted
  puts Base64.strict_decode64(converted)
  ```

  *produces:*

  ```
  Tm93IGlzIHRoZSBOaW1lIGZvciBhbGwgZ29vZCBjb2RlcnMKdG8gbGVhcm4gUnVieQ==
  Now is the time for all good coders
  to learn Ruby
  ```

Library

## Benchmark — Time Code Execution

Allows code execution to be timed and the results tabulated. The Benchmark module is easier to use if you include it in your top-level environment.

See also: Profile (page 792)

- Compares the costs of four kinds of method dispatch:

Download **samples/slbenchmark_1.rb**

```
require 'benchmark'
include Benchmark
string = "Stormy Weather"
m = string.method(:length)
bm(6) do |x|
  x.report("direct") { 100_000.times { string.length } }
  x.report("call")   { 100_000.times { m.call } }
  x.report("send")   { 100_000.times { string.send(:length) } }
  x.report("eval")   { 100_000.times { eval "string.length" } }
end
```

*produces:*

```
            user     system      total        real
direct  0.010000   0.000000   0.010000 (  0.011034)
call    0.020000   0.000000   0.020000 (  0.023135)
send    0.020000   0.000000   0.020000 (  0.016482)
eval    0.790000   0.000000   0.790000 (  0.800693)
```

- Which is better: reading all of a dictionary and splitting it or splitting it line by line? Use bmbm to run a rehearsal before doing the timing:

Download **samples/slbenchmark_2.rb**

```
require 'benchmark'
include Benchmark
bmbm(6) do |x|
  x.report("all") do
     str = File.read("/usr/share/dict/words")
     words = str.scan(/[-\w']+/)
  end
  x.report("lines") do
    words = []
    File.foreach("/usr/share/dict/words") do |line|
      words << line.chomp
    end
  end
end
```

*produces:*

```
Rehearsal -----------------------------------------
all     0.200000   0.010000   0.210000 (  0.220893)
lines   0.250000   0.010000   0.260000 (  0.259611)
-------------------------------- total: 0.470000sec

            user     system      total        real
all     0.220000   0.010000   0.230000 (  0.233112)
lines   0.220000   0.020000   0.240000 (  0.239560)
```

Library

## BigDecimal — Large-Precision Decimal Numbers

Ruby's standard Bignum class supports integers with large numbers of digits. The BigDecimal class supports decimal numbers with large numbers of decimal places. The standard library supports all the normal arithmetic operations. BigDecimal also comes with some extension libraries.

**bigdecimal/ludcmp**
: Performs an LU decomposition of a matrix.

**bigdecimal/math**
: Provides the transcendental functions $sqrt$, $sin$, $cos$, $atan$, $exp$, and $log$, along with functions for computing $PI$ and $E$. All functions take an arbitrary precision argument.

**bigdecimal/jacobian**
: Constructs the Jacobian (a matrix enumerating the partial derivatives) of a given function. Not dependent on BigDecimal.

**bigdecimal/newton**
: Solves the roots of nonlinear function using Newton's method. Not dependent on BigDecimal.

**bigdecimal/nlsolve**
: Wraps the bigdecimal/newton library for equations of BigDecimals.

You can find English-language documentation in the Ruby source distribution in the file `ext/bigdecimal/bigdecimal_en.html`.

Download **samples/slbigdecimal_1.rb**

```
require 'bigdecimal'
require 'bigdecimal/math'
include BigMath

pi = BigMath::PI(20)    # 20 is the number of decimal digits

radius = BigDecimal("2.14156987652974674392")

area = pi * radius**2

area.to_s              # =>   "0.144083540446856044176720033806679561688
                              599846410445032583215824758780405545861
                              780909930190528E2"

# The same with regular floats

radius = 2.14156987652974674392

Math::PI * radius**2   # =>   14.4083540446856
```

Library

## CGI — CGI Programming Support

The CGI class provides support for programs used as Common Gateway Interface (CGI) scripts in a web server. CGI objects are initialized with data from the environment and from the HTTP request, and they provide convenient accessors to form data and cookies. They can also manage sessions using a variety of storage mechanisms. Class CGI also provides basic facilities for HTML generation and class methods to escape and unescape requests and HTML.

See also: CGI::Session (page 735)

1.9

- Escapes and unescapes special characters in URLs and HTML. Numeric entities below 256 will be encoded based on the encoding of the input string. Other numeric entities will be left unchanged.

Download samples/slcgi_1.rb

```
require 'cgi'
CGI.escape('c:\My Files')         # =>   c%3A%5CMy+Files
CGI.unescape('c%3a%5cMy+Files')   # =>   c:\My Files
CGI::escapeHTML('"a"<b & c')      # =>   &quot;a&quot;&lt;b &amp; c

CGI.unescapeHTML('&quot;a&quot;&lt;=&gt;b')   # =>   "a"<=>b
CGI.unescapeHTML('&#65;&#x41;')               # =>   AA
str = '&#x3c0;r&#178;'
str.force_encoding("utf-8")
CGI.unescapeHTML(str)                         # =>   πr²
```

- Accesses information from the incoming request:

Download samples/slcgi_3.rb

```
require 'cgi'
c = CGI.new
c.auth_type    # =>   "basic"
c.user_agent   # =>   "Mozscape Explorari V5.6"
```

- Accesses form fields from an incoming request. Assume that the following script is installed as `test.cgi` and the user linked to it using `http://mydomain.com/test.cgi?fred=10&barney=cat`:

Download samples/slcgi_4.rb

```
require 'cgi'
c = CGI.new
c['fred']   # =>   "10"
c.keys      # =>   ["fred", "barney"]
c.params    # =>   {"fred"=>["10"], "barney"=>["cat"]}
```

- If a form contains multiple fields with the same name, the corresponding values will be returned to the script as an array. The [] accessor returns just the first of these—index the result of the params method to get them all.

In this example, assume the form has three fields called “name”:

Download samples/slcgi_5.rb

```
require 'cgi'
c = CGI.new
c['name']          # =>   "fred"
c.params['name']   # =>   ["fred", "wilma", "barney"]
c.keys             # =>   ["name"]
c.params           # =>   {"name"=>["fred", "wilma", "barney"]}
```

- Sends a response to the browser. (Not many folks use this form of HTML generation. Consider one of the templating libraries—see page 308.)

Download samples/slcgi_6.rb

```
require 'cgi'
cgi = CGI.new("html4Tr")
cgi.header("type" => "text/html", "expires" => Time.now + 30)
cgi.out do
  cgi.html do
    cgi.head{ cgi.title{"Hello World!"} } +
    cgi.body do
      cgi.pre do
        CGI::escapeHTML(
         "params: " + cgi.params.inspect + "\n" +
         "cookies: " + cgi.cookies.inspect + "\n")
      end
    end
  end
end
```

- Stores a cookie in the client browser:

Download samples/slcgi_7.rb

```
require 'cgi'
cgi = CGI.new("html4")
cookie = CGI::Cookie.new('name' => 'mycookie',
                         'value' => 'chocolate chip',
                         'expires' => Time.now + 3600)
cgi.out('cookie' => cookie) do
  cgi.head + cgi.body { "Cookie stored" }
end
```

- Retrieves a previously stored cookie:

Download samples/slcgi_8.rb

```
require 'cgi'
cgi = CGI.new("html4")
cookie = cgi.cookies['mycookie']
cgi.out('cookie' => cookie) do
  cgi.head + cgi.body { "Flavor: " + cookie[0] }
end
```

Library

## CGI::Session — CGI Sessions

A CGI::Session maintains a persistent state for web users in a CGI environment. Sessions may be memory resident or may be stored on disk. See the discussion on page 313 for details.

See also: CGI (page 733)

Download samples/slcgisession_1.rb

```
require 'cgi'
require 'cgi/session'
cgi = CGI.new("html3")
sess = CGI::Session.new(cgi,
                        "session_key" => "rubyweb",
                        "prefix" => "web-session.")
if sess['lastaccess']
  msg = "<p>You were last here #{sess['lastaccess']}.</p>"
else
  msg = "<p>Looks like you haven't been here for a while</p>"
end
count = (sess["accesscount"] || 0).to_i
count += 1
msg << "<p>Number of visits: #{count}</p>"
sess["accesscount"] = count
sess["lastaccess"]  = Time.now.to_s
sess.close
cgi.out {
  cgi.html {
    cgi.body {
      msg
    }
  }
}
```

Library **CMath** Complex Transcendental Functions

1.9

As of Ruby 1.9, Complex class is built in to the interpreter. There is no need to require the complex library to create and manipulate complex numbers. However, if you want the transcendental functions defined by the Math to work with complex numbers, you must also require the cmath library. The functions affected are as follows: acosh, acos, asinh, asin, atan2, atanh, atan, cosh, cos, exp, log10, log, sinh, sin, sqrt, tanh, and tan.

The Complex library makes these complex functions the default (so, if you require 'complex', you can use Math::sin and not CMath::sin).

Download **samples/slcmath_1.rb**

```
require 'cmath'
point = Complex(2, 3)
CMath::sin(point)   # =>   (9.15449914691143-4.16890695996656i)
CMath::cos(point)   # =>   (-4.18962569096881-9.10922789375534i)
```

Library **Complex** Complex Numbers

1.9

Loads the cmath library, which defines the transcendental functions for complex numbers. It then arranges things so that these complex-aware functions are the ones invoked when you use Math::. The net effect is that, after requiring complex, you can use functions such as Math::sin on any numeric value, including complex numbers.

Using transcendental numbers with complex arguments will, by default, cause an error:

Download **samples/slcomplex_1.rb**

```
point = Complex(2, 3)
Math::sin(point)
```

*produces:*

```
prog.rb:2:in `to_f': can't convert 2+3i into Float (RangeError)
from /tmp/prog.rb:2:in `sin'
from /tmp/prog.rb:2:in `<main>'
```

However...

Download **samples/slcomplex_2.rb**

```
require 'complex'
point = Complex(2, 3)
Math::sin(point)   # =>   (9.15449914691143-4.168906959966656i)
```

Library

## Continuation — Continuations

Continuation objects are generated by the Kernel#callcc method, which becomes available only when the continuation library is loaded. They hold a return address and execution context, allowing a nonlocal return to the end of the callcc block from anywhere within a program. Continuations are somewhat analogous to a structured version of C's setjmp/longjmp (although they contain more state, so you may consider them closer to threads). This (somewhat contrived) example allows the inner loop to abandon processing early.

- Does a nonlocal exit when a condition is met:

Download **samples/slcontinuation_1.rb**

```
require 'continuation'
callcc do |cont|
  for i in 0..4
    print "\n#{i}: "
    for j in i*5...(i+1)*5
      cont.call() if j == 7
      printf "%3d", j
    end
  end
end
print "\n"
```

*produces:*

```
0:   0  1  2  3  4
1:   5  6
```

- The call stack for methods is preserved in continuations:

Download **samples/slcontinuation_2.rb**

```
require 'continuation'
def strange
  callcc {|continuation| return continuation}
  print "Back in method, "
end
print "Before method. "
continuation = strange()
print "After method. "
continuation.call if continuation
```

*produces:*

```
Before method. After method. Back in method, After method.
```

Library

## CSV — Comma-Separated Values

1.9

Comma-separated data files are often used to transfer tabular information (and are a *lingua franca* for importing and exporting spreadsheet and database information). As of Ruby 1.9, the old library has been replaced by James Edward Gray II's FasterCSV version. It has a few incompatibilities with the original. In particular, CSV.open now works like File.open, not File.foreach, and options are passed as a hash and not positional parameters.

Ruby's CSV library deals with arrays (corresponding to the rows in the CSV file) and strings (corresponding to the elements in a row). If an element in a row is missing, it will be represented as a nil in Ruby.

The files used in the following examples are as follows:

**csvfile:**

```
12,eggs,2.89,
2,"shirt, blue",21.45,special
1,"""Hello Kitty"" bag",13.99
```

**csvfile_hdr:**

```
Count,Description,Price
12,eggs,2.89,
2,"shirt, blue",21.45,special
1,"""Hello Kitty"" bag",13.99
```

- Reads a file containing CSV data and process line by line:

  Download **samples/slcsv_1.rb**

  ```
  require 'csv'
  CSV.foreach("csvfile") do |row|
    qty = row[0].to_i
    price = row[2].to_f
    printf "%20s: $%5.2f %s\n", row[1], qty*price, row[3] || "  ---"
  end
  ```

  *produces:*

  ```
                  eggs: $34.68   ---
           shirt, blue: $42.90 special
    "Hello Kitty" bag: $13.99   ---
  ```

- Processes a CSV file that contains a header line. Automatically converts fields that look like numbers.

  Download **samples/slcsv_2.rb**

  ```
  require 'csv'
  total_cost = 0
  CSV.foreach("csvfile_hdr", headers: true, converters: :numeric)  do |data|
     total_cost += data["Count"] * data["Price"]
  end
  puts "Total cost is #{total_cost}"
  ```

  *produces:*

  ```
  Total cost is 91.57
  ```

- Writes CSV data to an existing open stream (STDOUT in this case). Uses | as the column separator.

Download **samples/slcsv_3.rb**

```
require 'csv'
CSV(STDOUT, col_sep: "|") do |csv|
  csv << [ 1, "line 1", 27 ]
  csv << [ 2, nil, 123 ]
  csv << [ 3, "|bar|", 32.5]
end
```

*produces:*

```
1|line 1|27
2||123
3|"|bar|"|32.5
```

- You can access a CSV file as a two-dimensional table:

Download **samples/slcsv_4.rb**

```
require 'csv'
table = CSV.read("csvfile_hdr",
                 headers: true,
                 header_converters: :symbol)
puts "Row count = #{table.count}"
puts "First row = #{table[0].fields}"
puts "Count of eggs = #{table[0][:count]}"
table << [99, "red balloons", 1.23]
table[:in_stock] = [10, 5, 10, 10]
puts "\nAfter adding a row and a column, the new table is:"
puts table
```

*produces:*

```
Row count = 3
First row = ["12", "eggs", "2.89", nil]
Count of eggs = 12

After adding a row and a column, the new table is:
count,description,price,,in_stock
12,eggs,2.89,,10
2,"shirt, blue",21.45,special,5
1,"""Hello Kitty"" bag",13.99,10
99,red balloons,1.23,,10
```

Library

## Curses — CRT Screen Handling



**Only if:** curses or ncurses installed in target environment

The Curses library is a fairly thin wrapper around the C curses or ncurses libraries, allowing applications a device-independent way of drawing on consoles and other terminal-like devices. As a nod toward object-orientation, curses windows and mouse events are represented as Ruby objects. Otherwise, the standard curses calls and constants are simply defined in the Curses module.

Download **samples/slcurses_1.rb**

```
# Draw the paddle of a simple game of 'pong'. It moves
# in response to the up and down keys
require 'curses'
include Curses
class Paddle
  HEIGHT = 4
  PADDLE = " \n" + "|\n"*HEIGHT + " "
  def initialize
    @top = (Curses::lines - HEIGHT)/2
    draw
  end
  def up
    @top -= 1 if @top > 1
  end
  def down
    @top += 1 if (@top + HEIGHT + 1) < lines
  end
  def draw
    setpos(@top-1, 0)
    addstr(PADDLE)
    refresh
  end
end
init_screen
begin
  crmode
  noecho
  stdscr.keypad(true)
  paddle = Paddle.new
  loop do
    case ch = getch
    when "Q".ord, "q".ord then   break
    when Key::UP          then   paddle.up
    when Key::DOWN        then   paddle.down
    else beep
    end
    paddle.draw
  end
ensure
  close_screen
end
```

Library

## Date/DateTime — Date and Time Manipulation



The date library implements classes Date and DateTime, which provide a comprehensive set of facilities for storing, manipulating, and converting dates with or without time components. The classes can represent and manipulate civil, ordinal, commercial, Julian, and standard dates, starting January 1, 4713 BCE. The DateTime class extends Date with hours, minutes, seconds, and fractional seconds, and it provides some support for time zones. The classes also provide support for parsing and formatting date and datetime strings. The classes have a rich interface—consult the ri documentation for details. The introductory notes in the file `lib/date.rb` are also well worth reading.

- Experiment with various representations:

Download **samples/sldate_1.rb**

```
require 'date'

d = Date.new(2000, 3, 31)
[d.year, d.yday, d.wday]       # =>  [2000, 91, 5]
[d.month, d.mday]              # =>  [3, 31]
[d.cwyear, d.cweek, d.cwday]   # =>  [2000, 13, 5]
[d.jd, d.mjd]                  # =>  [2451635, 51634]
d1 = Date.commercial(2000, 13, 7)
d1.to_s                        # =>  "2000-04-02"
[d1.cwday, d1.wday]            # =>  [7, 0]
```

- Essential information about Christmas:

Download **samples/sldate_2.rb**

```
require 'date'
now = DateTime.now
year = now.year
year += 1 if now.month == 12 && now.day > 25
xmas = DateTime.new(year, 12, 25)
diff = xmas - now
puts "It's #{diff.to_i} days to Christmas"
puts "Christmas #{year} falls on a #{xmas.strftime('%A')}"
```

*produces:*

```
It's 255 days to Christmas
Christmas 2009 falls on a Friday
```

Library

# DBM — Interface to DBM Databases

**Only if:** a DBM library is installed in target environment

DBM files implement simple, hashlike persistent stores. Many DBM implementations exist: the Ruby library can be configured to use one of the DBM libraries db, dbm (ndbm), gdbm, and qdbm. The interface to DBM files is similar to class Hash, except that DBM keys and values will be strings. This can cause confusion, because the conversion to a string is performed silently when the data is written. The DBM library is a wrapper around the lower-level access method. For true low-level access, see also the GDBM and SDBM libraries.

See also: gdbm (page 758), sdbm (page 806)



- Creates a simple DBM file and then reopens it read-only and reads some data. Note the conversion of a date object to its string form.

Download **samples/sldbm_1.rb**

```
require 'dbm'
require 'date'
DBM.open("data.dbm") do |dbm|
  dbm['name'] = "Walter Wombat"
  dbm['dob']  = Date.new(1997, 12,25)
end
DBM.open("data.dbm", nil, DBM::READER) do |dbm|
  p dbm.keys
  p dbm['dob']
  p dbm['dob'].class
end
```

*produces:*

```
["name", "dob"]
"1997-12-25"
String
```

- Reads from the system's *aliases* file. Note the trailing null bytes on all strings.

Download **samples/sldbm_2.rb**

```
require 'dbm'
DBM.open("/etc/aliases", nil) do |dbm|
  p dbm.keys
  p dbm["postfix\000"]
end
```

*produces:*

```
["postmaster:\x00", "daemon:\x00", "ftp-bugs:\x00", "operator:\x00",
 "abuse:\x00", "decode:\x00", "mailer-daemon:\x00", "bin:\x00",
 "named:\x00", "nobody:\x00", "uucp:\x00", "www:\x00", "postfix:\x00",
 "manager:\x00", "dumper:\x00"]
nil
```

Library

## Delegator — Delegate Calls to Other Object

Object delegation is a way of *composing* objects—extending an object with the capabilities of another—at runtime. The Ruby Delegator class implements a simple but powerful delegation scheme, where requests are automatically forwarded from a master class to delegates or their ancestors and where the delegate can be changed at runtime with a single method call.

See also: Forwardable (page 757)

- For simple cases where the class of the delegate is fixed, make the master class a subclass of DelegateClass, passing the name of the class to be delegated as a parameter. In the master class's initialize method, pass the object to be delegated to the superclass.

Download **samples/sldelegate_1.rb**

```
require 'delegate'

class Words < DelegateClass(Array)
  def initialize(list = "/usr/share/dict/words")
    words = File.read(list).split
    super(words)
  end
end

words = Words.new
words[9999]          # =>   "anticritique"
words.size           # =>   234936
words.grep(/matz/)   # =>   ["matzo", "matzoon", "matzos", "matzoth"]
```

- Use SimpleDelegator to delegate to a particular object (which can be changed):

Download **samples/sldelegate_2.rb**

```
require 'delegate'

words = File.read("/usr/share/dict/words").split
names = File.read("/usr/share/dict/propernames").split

stats = SimpleDelegator.new(words)
stats.size   # =>   234936
stats[226]   # =>   "abidingly"
stats.__setobj__(names)
stats.size   # =>   1323
stats[226]   # =>   "Dave"
```

Library

## Digest — MD5, RIPEMD-160 SHA1, and SHA2 Digests

The Digest module is the home for a number of classes that implement message digest algorithms: MD5, RIPEMD-160, SHA1, and SHA2 (256, 384, and 512 bit). The interface to all these classes is identical.

- You can create a binary or hex digest for a given string by calling the class method digest or hexdigest.
- You can also create an object (optionally passing in an initial string) and determine the object's hash by calling the digest or hexdigest instance methods. In this case, you can then append to the string using the update method and then recover an updated hash value.
- Calculates some MD5 and SHA1 hashes:

Download **samples/sldigest_1.rb**

```
require 'digest/md5'
require 'digest/sha1'
for hash_class in [ Digest::MD5, Digest::SHA1 ]
  puts "Using #{hash_class.name}"
  # Calculate directly
  puts hash_class.hexdigest("hello world")
  # Or by accumulating
  digest = hash_class.new
  digest << "hello"
  digest << " "
  digest << "world"
  puts digest.hexdigest
  puts
end
```

*produces:*

```
Using Digest::MD5
5eb63bbbe01eeed093cb22bb8f5acdc3
5eb63bbbe01eeed093cb22bb8f5acdc3

Using Digest::SHA1
2aae6c35c94fcfb415dbe95f408b9ce91ee846ed
2aae6c35c94fcfb415dbe95f408b9ce91ee846ed
```

Library

## DL — Access Dynamically Loaded Libraries (.dll and .so)

**Only if:** Windows, or system supports dl library

The DL module interfaces to the underlying operating system's dynamic loading capabilities. On Windows boxes, it can be used to interface with functions in DLLs. Under Unix it can load shared libraries. Because Ruby does not have typed method parameters or return values, you must define the types expected by the methods you call by specifying their signatures. This can be done using a C-like syntax (if you use the high-level methods in dl/import) or using explicit type specifiers in the lower-level DL module. Good documentation is provided in the source tree's ext/dl/doc/ directory.

- Here's a trivial C program that we'll build as a shared library:

  Download **samples/sldl_1.rb**

```
#include <stdio.h>
int print_msg(text, number) {
  return printf("Text: %s (%d)\n", text, number);
}
```

- Generates a proxy to access the print_msg method in the shared library. The way this book is built, the shared library is in the subdirectory `code/dl`; this directory must be added to the directories searched when looking for dynamic objects.

  Download **samples/sldl_2.rb**

```
ENV['DYLD_LIBRARY_PATH'] = ":code/dl"  # Mac OS X
require 'dl/func'
lib = DL.dlopen("code/dl/lib.so")
cfunc = DL::CFunc.new(lib['print_msg'], DL::TYPE_INT, 'print_msg')
print_msg = DL::Function.new(cfunc, [DL::TYPE_VOIDP, DL::TYPE_INT])
msg_size = print_msg.call("Answer", 42)
puts "Just wrote #{msg_size} bytes"
```

  *produces:*

```
Just wrote 18 bytes
Text: Answer (42)
```

- We can also wrap the method in a module:

  Download **samples/sldl_3.rb**

```
ENV['DYLD_LIBRARY_PATH'] = ":code/dl"  # Mac OS X
require 'dl/import'
module Message
  extend DL::Importer
  dlload "lib.so"
  extern "int print_msg(char *, int)"
end
msg_size = Message.print_msg("Answer", 42)
puts "Just wrote #{msg_size} bytes"
```

  *produces:*

```
Just wrote 18 bytes
Text: Answer (42)
```

Library

## dRuby — Distributed Ruby Objects (drb)

dRuby allows Ruby objects to be distributed across a network connection. Although expressed in terms of clients and servers, once the initial connection is established, the protocol is effectively symmetrical: either side can invoke methods in objects on the other side. Normally, objects passed and returned by remote calls are passed by value; including the DRbUndumped module in an object forces it to be passed by reference (useful when implementing callbacks).

See also: Rinda (page 801), XMLRPC (page 830)

- This server program is *observable*—it notifies all registered listeners of changes to a count value:

Download **samples/sldrb_1.rb**

```
require 'drb'
require 'drb/observer'
class Counter
  include DRb::DRbObservable
  def run
    5.times do |count|
      changed
      notify_observers(count)
    end
  end
end
counter = Counter.new
DRb.start_service('druby://localhost:9001', counter)
DRb.thread.join
```

- This client program interacts with the server, registering a listener object to receive callbacks before invoking the server's run method:

Download **samples/sldrb_2.rb**

```
require 'drb'
class Listener
  include DRbUndumped
  def update(value)
    puts value
  end
end
DRb.start_service
counter = DRbObject.new(nil, "druby://localhost:9001")
listener = Listener.new
counter.add_observer(listener)
counter.run
```

Library

## English — English Names for Global Symbols

Includes the English library file in a Ruby script, and you can reference the global variables such as $_ using less-cryptic names, listed in the following table.English. It is now predefined in the Ruby interpreter.

| | | | |
|---|---|---|---|
| $* | $ARGV | $_ | $LAST_READ_LINE |
| $? | $CHILD_STATUS | $" | $LOADED_FEATURES |
| $< | $DEFAULT_INPUT | $& | $MATCH |
| $> | $DEFAULT_OUTPUT | $. | $NR |
| $! | $ERROR_INFO | $, | $OFS |
| $@ | $ERROR_POSITION | $\ | $ORS |
| $; | $FIELD_SEPARATOR | $, | $OUTPUT_FIELD_SEPARATOR |
| $; | $FS | $\ | $OUTPUT_RECORD_SEPARATOR |
| $= | $IGNORECASE | $$ | $PID |
| $. | $INPUT_LINE_NUMBER | $' | $POSTMATCH |
| $/ | $INPUT_RECORD_SEPARATOR | $` | $PREMATCH |
| $~ | $LAST_MATCH_INFO | $$ | $PROCESS_ID |
| $+ | $LAST_PAREN_MATCH | $/ | $RS |

Download **samples/slenglish_1.rb**

```
require 'English'
$OUTPUT_FIELD_SEPARATOR = ' -- '
"waterbuffalo" =~ /buff/
print $., $INPUT_LINE_NUMBER, "\n"
print $', $POSTMATCH, "\n"
print $$, $PID
```

*produces:*

```
0 -- 0 --
 -- alo -- alo --
 -- 86006 -- 86006 --
```

Library

## erb — Lightweight Templating for HTML

ERb is a lightweight templating system, allowing you to intermix Ruby code and plain text. This is sometimes a convenient way to create HTML documents but also is usable in other plain-text situations. For other templating solutions, see 308.

ERB breaks its input text into chunks of regular text and program fragments. It then builds a Ruby program that, when run, outputs the result text and executes the program fragments. Program fragments are enclosed between <% and %> markers. The exact interpretation of these fragments depends on the character following the opening <%, as shown in Table 28.1 on the following page.

Download **samples/slerb_1.rb**

```
require 'erb'
input = %{<% high.downto(low) do |n|   # set high, low externally %>
  <%= n %> green bottles, hanging on the wall
  <%= n %> green bottles, hanging on the wall
  And if one green bottle should accidentally fall
  There'd be <%= n-1 %> green bottles, hanging on the wall
<% end %>}
high,low = 10, 8
erb = ERB.new(input)
erb.run(binding)
```

*produces:*

```
  10 green bottles, hanging on the wall
  10 green bottles, hanging on the wall
  And if one green bottle should accidentally fall
  There'd be 9 green bottles, hanging on the wall
         . . .
```

An optional second parameter to ERB.new sets the safe level for evaluating expressions. If nil, expressions are evaluated in the current thread; otherwise, a new thread is created, and its $SAFE level is set to the parameter value.

The optional third parameter to ERB.new allows some control of the interpretation of the input and of the way whitespace is added to the output. If the third parameter is a string and that string contains a percent sign, then ERB treats lines starting with a percent sign specially. Lines starting with a single percent sign are treated as if they were enclosed in <%...%>. Lines starting with a double percent sign are copied to the output with a single leading percent sign.

```
str = %{\
% 2.times do |i|
  This is line <%= i %>
%end
%% done}
ERB.new(str, 0, '%').run
```

⇒ *produces:*

```
  This is line 0
  This is line 1
% done
```

If the third parameter contains the string <>, then a newline will not be written if an input line starts with an ERB directive and ends with %>. If the trim parameter contains >, then a newline will not be written if an input line ends %>.

Table 28.1. Directives for ERB

| Sequence | Action |
|---|---|
| <% *ruby code* %> | Inserts the given Ruby code at this point in the generated program. If it outputs anything, include this output in the result. |
| <%= *ruby expression* %> | Evaluate expression and insert its value in the output of the generated program. |
| <%# . . . %> | Comment (ignored). |
| <%% and %%> | Replaced in the output by <% and%> respectively. |



Download **samples/slerb_4.rb**

```
str1 = %{\
* <%= "cat" %>
<%= "dog" %>
}
ERB.new(str1, 0, ">").run
ERB.new(str1, 0, "<>").run
```

*produces:*

```
* catdog* cat
dog
```

The erb library also defines the helper module ERB::Util that contains two methods: html_escape (aliased as h) and url_encode (aliased as u). These are equivalent to the CGI methods escapeHTML and escape, respectively (except escape encodes spaces as plus signs, and url_encode uses %20).

Download **samples/slerb_5.rb**

```
include ERB::Util
str1 = %{\
h(a) = <%= h(a) %>
u(a) = <%= u(a) %>
}
a = "< a & b >"
ERB.new(str1).run(binding)
```

*produces:*

```
h(a) = &lt; a &amp; b &gt;
u(a) = %3C%20a%20%26%20b%20%3E
```

You may find the command-line utility erb is supplied with your Ruby distribution. This allows you to run erb substitutions on an input file; see erb --help for details.

Library

## Etc — Access User and Group Information in /etc/passwd

**Only if:** Unix or Cygwin

The Etc module provides a number of methods for querying the passwd and group facilities on Unix systems.

- Finds out information about the currently logged-in user:

Download **samples/sletc_1.rb**

```
require 'etc'

name = Etc.getlogin
info = Etc.getpwnam(name)
info.name    # =>   "dave"
info.uid     # =>   501
info.dir     # =>   "/Users/dave"
info.shell   # =>   "/bin/bash"

group = Etc.getgrgid(info.gid)
group.name   # =>   "dave"
```

- Returns the names of users on the system used to create this book:

Download **samples/sletc_2.rb**

```
require 'etc'

users = []
Etc.passwd {|passwd| users << passwd.name }
users[1,5].join(", ")   # =>   "_appowner, _appserver, _ard,
                                _atsserver, _calendar"
```

- Returns the IDs of groups on the system used to create this book:

Download **samples/sletc_3.rb**

```
require 'etc'

ids = []
Etc.group {|entry| ids << entry.gid }
ids[1,5].join(", ")   # =>   "87, 81, 79, 67, 97"
```

Library

## expect — Expect Method for IO Objects

The expect library adds the method expect to all IO objects. This allows you to write code that waits for a particular string or pattern to be available from the I/O stream. The expect method is particularly useful with pty objects (see page 795) and with network connections to remote servers, where it can be used to coordinate the use of external interactive processes.

If the global variable $expect_verbose is true, the expect method writes all characters read from the I/O stream to STDOUT.

See also: pty (page 795)

- Connects to the local FTP server, logs in, and prints out the name of the user's directory. (Note that it would be a lot easier to do this using the net/ftp library.)

Download **samples/slexpect_1.rb**

```
# This code might be specific to the particular
# ftp daemon.
require 'expect'
require 'socket'
$expect_verbose = true
socket = TCPSocket.new('localhost', 'ftp')
socket.expect("ready")
socket.puts("user testuser")
socket.expect("Password required for testuser")
socket.puts("pass secret")
socket.expect("logged in.\r\n")
socket.puts("pwd")
puts(socket.gets)
socket.puts "quit"
```

*produces:*

```
220 localhost FTP server (tnftpd 20061217) ready.
331 Password required for testuser.
230 User testuser logged in.
257 "/Users/testuser" is the current directory.
```

Library

## Fcntl — Symbolic Names for IO#fcntl Commands

The Fcntl module provides symbolic names for each of the host system's available fcntl constants (defined in fcntl.h). That is, if the host system has a constant named F_GETLK defined in fcntl.h, then the Fcntl module will have a corresponding constant Fcntl::F_GETLK with the same value as the header file's #define.

- Different operating system will have different Fcntl constants available. The value associated with a constant of a given name may also differ across platforms. Here are the values on my Mac OS X system:

Download **samples/slfcntl_1.rb**

```
require 'fcntl'
Fcntl.constants.sort.each do |name|
  printf "%10s: 0x%06x\n", name, Fcntl.const_get(name)
end
```

*produces:*

```
FD_CLOEXEC: 0x000001
   F_DUPFD: 0x000000
   F_GETFD: 0x000001
   F_GETFL: 0x000003
   F_GETLK: 0x000007
   F_RDLCK: 0x000001
   F_SETFD: 0x000002
   F_SETFL: 0x000004
   F_SETLK: 0x000008
  F_SETLKW: 0x000009
   F_UNLCK: 0x000002
   F_WRLCK: 0x000003
 O_ACCMODE: 0x000003
  O_APPEND: 0x000008
   O_CREAT: 0x000200
    O_EXCL: 0x000800
  O_NDELAY: 0x000004
  O_NOCTTY: 0x020000
O_NONBLOCK: 0x000004
  O_RDONLY: 0x000000
    O_RDWR: 0x000002
   O_TRUNC: 0x000400
  O_WRONLY: 0x000001
```

Library

## Fiber — Coroutines Using Fibers

The Fiber class that is built into Ruby provides a generator-lke capability—fibers may be created and resumed from some controlling program. If you want to extend the Fiber class to provide full, symmetrical coroutines, you need first to require the fiber library. This adds two instance methods, transfer and alive? to Fiber objects, and the singleton method current to the Fiber class.

- It is difficult to come up with a meaningful, concise example of symmetric coroutines that can't more easily be coded with asymetric (plain old) fibers. So, here's an artificial example....

Download **samples/slfiber_1.rb**

```
require 'fiber'
# take items two at a time off a queue, calling the producer
# if not enough are available
consumer = Fiber.new do |producer, queue|
  5.times do
    while queue.size < 2
      queue = producer.transfer(consumer, queue)
    end
    puts "Consume #{queue.shift} and #{queue.shift}"
  end
end
# add items three at a time to the queue
producer = Fiber.new do |consumer, queue|
  value = 1
  loop do
    puts "Producing more stuff"
    3.times { queue << value; value += 1}
    puts "Queue size is #{queue.size}"
    consumer.transfer queue
  end
end
consumer.transfer(producer, [])
```

*produces:*

```
Producing more stuff
Queue size is 3
Consume 1 and 2
Producing more stuff
Queue size is 4
Consume 3 and 4
Consume 5 and 6
Producing more stuff
Queue size is 3
Consume 7 and 8
Producing more stuff
Queue size is 4
Consume 9 and 10
```

Library

## FileUtils — File and Directory Manipulation

FileUtils is a collection of methods for manipulating files and directories. Although generally applicable, the model is particularly useful when writing installation scripts.

Many methods take a *src* and a *dest* parameter. If *dest* is a directory, *src* may be a single filename or an array of filenames. For example, the following copies the files a, b, and c to /tmp:

```
cp( %w{ a b c }, "/tmp")
```

Most functions take a set of options. These may be zero or more of the following:

| Option | Meaning |
|---|---|
| :verbose | Traces execution of each function (by default to STDERR, although this can be overridden by setting the class variable @fileutils_output). |
| :noop | Does not perform the action of the function (useful for testing scripts). |
| :force | Overrides some default conservative behavior of the method (for example, overwriting an existing file). |
| :preserve | Attempts to preserve atime, mtime, and mode information from *src* in *dest*. (Setuid and setgid flags are always cleared.) |

For maximum portability, use forward slashes to separate the directory components of filenames, even on Windows.

FileUtils contains three submodules that duplicate the top-level methods but with different default options: module FileUtils::Verbose sets the verbose option, module FileUtils::NoWrite sets noop, and FileUtils::DryRun sets verbose and noop.

1.9

See also: un (page 825)

Download samples/slfileutils_2.rb

```
require 'fileutils'
include FileUtils::Verbose
cd("/tmp") do
  cp("/etc/passwd", "tmp_passwd")
  chmod(0666, "tmp_passwd")
  cp_r("/usr/include/net/", "headers")
  rm("tmp_passwd")     # Tidy up
  rm_rf("headers")
end
```

*produces:*

```
cd /tmp
cp /etc/passwd tmp_passwd
chmod 666 tmp_passwd
cp -r /usr/include/net/ headers
rm tmp_passwd
rm -rf headers
cd -
```

Library

## Find — Traverse Directory Trees

1.9

The Find module supports the top-down traversal of a set of file paths, given as arguments to the find method. If an argument is a file, its name is passed to the block associated with the call. If it's a directory, then its name and the name of all its files and subdirectories will be passed in. If no block is associated with the call, an Enumerator is returned.

Within the block, the method prune may be called, which skips the current file or directory, restarting the loop with the next directory. If the current file is a directory, that directory will not be recursively entered. In the following example, we don't list the contents of the local Subversion cache directories:

Download **samples/slfind_1.rb**

```
require 'find'
Find.find("/etc/passwd", "code/cdjukebox") do |f|
  type = case
         when File.file?(f)      then "File: "
         when File.directory?(f) then "Dir:  "
         else "?"
         end
  puts "#{type} #{f}"
  Find.prune if f =~ /.svn/
end
```

*produces:*

```
File:  /etc/passwd
Dir:   code/cdjukebox
File:  code/cdjukebox/Makefile
File:  code/cdjukebox/libcdjukebox.a
File:  code/cdjukebox/cdjukebox.o
File:  code/cdjukebox/cdjukebox.h
File:  code/cdjukebox/cdjukebox.c
Dir:   code/cdjukebox/.svn
```

Library

## Forwardable — Object Delegation

Forwardable provides a mechanism to allow classes to delegate named method calls to other objects.

See also: Delegator (page 744)

- This simple symbol table uses a hash, exposing a subset of the hash's methods:

Download samples/slforwardable_1.rb

```
require 'forwardable'

class SymbolTable
  extend Forwardable
  def_delegator(:@hash, :[],  :lookup)
  def_delegator(:@hash, :[]=, :add)
  def_delegators(:@hash, :size, :has_key?)
  def initialize
    @hash = Hash.new
  end
end

st = SymbolTable.new
st.add('cat', 'feline animal')   # =>   "feline animal"
st.add('dog', 'canine animal')   # =>   "canine animal"
st.add('cow', 'bovine animal')   # =>   "bovine animal"

st.has_key?('cow')               # =>   true
st.lookup('dog')                 # =>   "canine animal"
```

- Forwards can also be defined for individual objects by extending them with the SingleForwardable module. It's hard to think of a good reason to use this feature, so here's a silly one:

Download samples/slforwardable_2.rb

```
require 'forwardable'
TRICKS = [ "roll over", "play dead" ]
dog = "rover"
dog.extend SingleForwardable
dog.def_delegator(:TRICKS, :each, :can)
dog.can do |trick|
  puts trick
end
```

*produces:*

```
roll over
play dead
```

Library

# GDBM — Interface to GDBM Database

**Only if:** gdbm library available

Interfaces to the gdbm database library.[1] Although the DBM library provides generic access to gdbm databases, it doesn't expose some features of the full gdbm interface. The GDBM library gives you access to underlying gdbm features such as the cache size, synchronization mode, reorganization, and locking. Only one process may have a GDBM database open for writing (unless locking is disabled).

See also: DBM (page 743), SDBM (page 806)



- Stores some values into a database and then reads them back. The second parameter to the open method specifies the file mode, and the next parameter uses two flags that (1) create the database if it doesn't exist, and (2) force all writes to be synced to disk. Create on open is the default Ruby gdbm behavior.

  Download **samples/slgdbm_1.rb**

```
require 'gdbm'
GDBM.open("data.dbm", 0644, GDBM::WRCREAT | GDBM::SYNC) do |dbm|
  dbm['name'] = "Walter Wombat"
  dbm['dob']  = "1969-12-25"
  dbm['uses'] = "Ruby"
end
GDBM.open("data.dbm") do |dbm|
  p dbm.keys
  p dbm['dob']
  dbm.delete('dob')
  p dbm.keys
end
```

  *produces:*

```
["uses", "dob", "name"]
"1969-12-25"
["uses", "name"]
```

- Opens a database read-only. Note that the attempt to delete a key fails.

  Download **samples/slgdbm_2.rb**

```
require 'gdbm'
GDBM.open("data.dbm", 0, GDBM::READER) do |dbm|
  p dbm.keys
  dbm.delete('name')
end
```

  *produces:*

```
["uses", "name"]
prog.rb:4:in `delete': Reader can't delete (GDBMError)
from /tmp/prog.rb:5:in `block in <main>'
from /tmp/prog.rb:3:in `open'
```

1. http://www.gnu.org/software/gdbm/gdbm.html

Library

## GetoptLong — Parse Command-Line Options

Class GetoptLong supports GNU-style command-line option parsing. Options may be a minus sign (–) followed by a single character or may be two minus signs (- -) followed by a name (a long option). Long options may be abbreviated to their shortest unambiguous lengths.

A single internal option may have multiple external representations. For example, the option to control verbose output could be any of -v, --verbose, or --details. Some options may also take an associated value.

Each internal option is passed to GetoptLong as an array, containing strings representing the option's external forms and a flag. The flag specifies how GetoptLong is to associate an argument with the option (NO_ARGUMENT, REQUIRED_ARGUMENT, or OPTIONAL_ARGUMENT).

If the environment variable POSIXLY_CORRECT is set, all options must precede non-options on the command line. Otherwise, the default behavior of GetoptLong is to reorganize the command line to put the options at the front. This behavior may be changed by setting GetoptLong#ordering= to one of the constants PERMUTE, REQUIRE_ORDER, or RETURN_IN_ORDER. POSIXLY_CORRECT may not be overridden.

See also: OptionParser (page 785)

Download samples/slgetoptlong_1.rb

```
# Call using "ruby example.rb --size 10k -v -q a.txt b.doc"
require 'getoptlong'
# specify the options we accept and initialize
# the option parser
opts = GetoptLong.new(
  [ "--size",    "-s",            GetoptLong::REQUIRED_ARGUMENT ],
  [ "--verbose", "-v",            GetoptLong::NO_ARGUMENT ],
  [ "--query",   "-q",            GetoptLong::NO_ARGUMENT ],
  [ "--check",   "--valid", "-c", GetoptLong::NO_ARGUMENT ]
)
# process the parsed options
opts.each do |opt, arg|
  puts "Option: #{opt}, arg #{arg.inspect}"
end
puts "Remaining args: #{ARGV.join(', ')}"
```

*produces:*

```
Option: --size, arg "10k"
Option: --verbose, arg ""
Option: --query, arg ""
Remaining args: a.txt, b.doc
```

Library

## GServer — Generic TCP Server

Simple framework for writing TCP servers. Subclasses the GServer class, sets the port (and potentially other parameters) in the constructor, and then implements a serve method to handle incoming requests.

GServer manages a thread pool for incoming connections, so your serve method may be running in multiple threads in parallel.

You can run multiple GServer copies on different ports in the same application.

- When a connection is made on port 2000, responds with the current time as a string. Terminates after handling three requests.

  Download **samples/slgserver_1.rb**

```
require 'gserver'
class TimeServer < GServer
  def initialize
    super(2000)
    @count = 3
  end
  def serve(client)
    client.puts Time.now
    @count -= 1
    stop if @count.zero?
  end
end
server = TimeServer.new
server.audit = true    # enable logging
server.start
server.join
```

- You can test this server by reading from *localhost* on port 2000. We use curl to do this—you could also use telnet:

```
% curl -s localhost:2000
```

*produces:*

```
2009-04-13 13:27:00 -0500
```

Library

## Iconv — Character Encoding Conversion

**Only if:** libiconv installed

The Iconv class is an interface to the Open Group's iconv library, which supports the translation of strings between character encodings. For a list of the supported encodings on your platform, see the iconv_open man pages for your system.

An Iconv object encapsulates a conversion descriptor, which in turn contains the information needed to convert from one encoding to another. The converter can be used multiple times, until closed.

The conversion method iconv can be called multiple times to convert input strings. At the end, it should be called with a nil argument to flush out any remaining output.

The new string transcoding functions in Ruby 1.9 make the basic Iconv functions redundant. However, Iconv has capabilities (such as transliteration) that are not part of the built-in Ruby functionality.



- Converts from ISO-8859-1 to UTF-16:

```
require 'iconv'
conv = Iconv.new("UTF-16", "ISO-8859-1")
result = conv.iconv("hello")
result << conv.iconv(nil)
result.dump   # =>   "\xFE\xFF\x00h\x00e\x00l\x00l\x00o"
```

- Does the same conversion using a class method. Note that we use Iconv.conv, which returns a single string, as opposed to Iconv.iconv, which returns an array of strings.

```
require 'iconv'
result = Iconv.conv("UTF-16", "ISO-8859-1", "hello")
result.dump   # =>   "\xFE\xFF\x00h\x00e\x00l\x00l\x00o"
```

- Converts *olé* from UTF-8 to ISO-8859-1:

```
require 'iconv'
result = Iconv.conv("ISO-8859-1", "UTF-8", "ol\303\251")
result.dump   # =>   "ol\xE9"
```

- Converts *olé* from UTF-8 to ASCII. This throws an exception, because ASCII doesn't have an *é* character.

```
require 'iconv'
result = Iconv.conv("ASCII", "UTF-8", "ol\303\251")
```

*produces:*

```
prog.rb:2:in `conv': "\xC3\xA9" (Iconv::IllegalSequence)
from /tmp/prog.rb:2:in `<main>'
```

- This time, converts to ASCII with transliteration, which shows approximations of missing characters:

```
require 'iconv'
result = Iconv.iconv("ASCII//TRANSLIT", "UTF-8", "ol\303\251")
result[0].dump   # =>   "ol'e"
```

Library

## IO/Wait — Check for Pending Data to Be Read

**Only if:** FIONREAD feature in ioctl(2)

Including the library io/wait adds the methods IO#nread, IO#ready?, and IO#wait to the standard IO class. These allow an IO object opened on a stream (not a file) to be queried to see whether data is available to be read without reading it and to wait for a given number of bytes to become available.

- Sets up a pipe between two processes and writes 10 bytes at a time into it. Periodically sees how much data is available.

Download **samples/sliowait_1.rb**

```
require 'io/wait'
reader, writer = IO.pipe
if (pid = fork)
  writer.close
  8.times do
    sleep 0.03
    len = reader.ready?
    if len
      puts "#{len} bytes available: #{reader.sysread(len)}"
    else
      puts "No data available"
    end
  end
  Process.waitpid(pid)
else
  reader.close
  5.times do |n|
    sleep 0.04
    writer.write n.to_s * 10
  end
  writer.close
end
```

*produces:*

```
No data available
10 bytes available: 0000000000
10 bytes available: 1111111111
No data available
10 bytes available: 2222222222
10 bytes available: 3333333333
10 bytes available: 4444444444
No data available
```

Library

## IPAddr — Represent and Manipulate IP Addresses

Class IPAddr holds and manipulates Internet Protocol (IP) addresses. Each address contains three parts: an address, a mask, and an address family. The family will typically be AF_INET for IPv4 and IPv6 addresses. The class contains methods for extracting parts of an address, checking for IPv4 compatible addresses (and IPv4-mapped IPv6 addresses), testing whether an address falls within a subnet and many other functions. It is also interesting in that it contains as data its own unit tests.

```
require 'ipaddr'

v4 = IPAddr.new('192.168.23.0/24')
v4            # =>   #<IPAddr: IPv4:192.168.23.0/ 255.255.255.0>
v4.mask(16)   # =>   #<IPAddr: IPv4:192.168.0.0/ 255.255.0.0>
v4.reverse    # =>   "0.23.168.192.in-addr.arpa"
v6 = IPAddr.new('3ffe:505:2::1')
v6            # =>   #<IPAddr:
                     IPv6:3ffe:0505:0002:0000:0000:0000:0000:0001/
                     ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff>
v6.mask(48)   # =>   #<IPAddr:
                     IPv6:3ffe:0505:0002:0000:0000:0000:0000:0000/
                     ffff:ffff:ffff:0000:0000:0000:0000:0000>

# the value for 'family' is OS dependent. This
# value is for OS X
v6.family     # =>   30

other = IPAddr.new("192.168.23.56")
v4.include?(other)   # =>   true
```

Library **irb** Interactive Ruby

The irb library is most commonly associated with the console command irb. However, you can also start an irb session from within your running application. A common technique is to trap a signal and start irb in the handler.

The following program sets up a signal handler that runs irb when the user hits ^C. The user can change the value of the instance variable @value. When they exit from irb, the original program continues to run with that new value.

Download **samples/slirb_1.rb**

```
require 'irb'
trap "INT" do
  IRB.start
end
count = 0
loop do
  count += 1
  puts count
  puts "Value = #{@value}" if defined? @value
  sleep 1
end
```

Here's a simple session using it:

```
$ ruby code/run_irb.rb
1
2
3
^C4
irb(main):001:0> @value = "wibble"
=> "wibble"
irb(main):002:0> exit
5
Value = wibble
6
Value = wibble
...
```

Library 1.9 **json** Generate and Parse JSON Format

JSON is a language-independent data interchange format based on key/value pairs (hashes in Ruby) and sequences of values (arrays in Ruby).[2] JSON is frequently used to exchange data between JavaScript running in browsers and server-based applications. JSON is not a general-purpose object marshaling format. Although you can add to_json methods to your own classes, you will lose interoperability. See also: yaml (page 831)

- Serializes a data structure into a string and write that to a file:

  Download samples/sljson_1.rb

```
require 'json'
data = { name: 'dave', address: [ 'tx', 'usa' ], age: 17 }
serialized = data.to_json
serialized   # =>   {"name":"dave","address":["tx","usa"],"age":17}
File.open("data", "w") {|f| f.puts serialized}
```

- Reads the serialized data from the file and reconstitute it:

  Download samples/sljson_2.rb

```
require 'json'
serialized = File.read("data")
data = JSON.parse(serialized)
data   # =>   {"name"=>"dave", "address"=>["tx", "usa"], "age"=>17}
```

- The methods j and jj convert their argument to JSON and write the result to STDOUT (jj prettyprints). This can be useful in irb.

  Download samples/sljson_3.rb

```
require 'json'
data = { name: 'dave', address: [ 'tx', 'usa' ], age: 17 }
puts "Regular"
j data
puts "Pretty"
jj data
```

*produces:*

```
Regular
{"name":"dave","address":["tx","usa"],"age":17}
Pretty
{
  "name": "dave",
  "address": [
    "tx",
    "usa"
  ],
  "age": 17
}
```

2. http://www.ietf.org/rfc/rfc4627.txt

Library

## Logger — Application Logging

Writes log messages to a file or stream. Supports automatic time- or size-based rolling of log files. Messages can be assigned severities, and only those messages at or above the logger's current reporting level will be logged.

- During development, you may want to see all messages:

```
require 'logger'
log = Logger.new(STDOUT)
log.level = Logger::DEBUG
log.datetime_format = "%H:%M:%S"
log.info("Application starting")
3.times do |i|
  log.debug("Executing loop, i = #{i}")
  temperature = some_calculation(i)  # defined externally
  if temperature > 50
    log.warn("Possible overheat. i = #{i}")
  end
end
log.info("Application terminating")
```

*produces:*

```
I, [13:27:00#86139]  INFO -- : Application starting
D, [13:27:00#86139] DEBUG -- : Executing loop, i = 0
D, [13:27:00#86139] DEBUG -- : Executing loop, i = 1
D, [13:27:00#86139] DEBUG -- : Executing loop, i = 2
W, [13:27:00#86139]  WARN -- : Possible overheat. i = 2
I, [13:27:00#86139]  INFO -- : Application terminating
```

- In deployment, you can turn off anything below INFO:

```
require 'logger'
log = Logger.new(STDOUT)
log.level = Logger::INFO
log.datetime_format = "%H:%M:%S"
# as above...
```

*produces:*

```
I, [13:27:00#86141]  INFO -- : Application starting
W, [13:27:00#86141]  WARN -- : Possible overheat. i = 2
I, [13:27:00#86141]  INFO -- : Application terminating
```

- Logs to a file, which is rotated when it gets to about 10KB. Keeps up to five old files.

```
require 'logger'
log = Logger.new("application.log", 5, 10*1024)
log.info("Application starting")
# ...
```

Library

# mathn — Unified Numbers

The mathn library attempts to bring some unity to numbers under Ruby, making classes Bignum, Complex, Fixnum, Integer, and Rational work and play better together. It automatically includes the libraries complex, rational, matrix, and prime.

- Types will tend to convert between themselves in a more natural way (so, for example, Complex::I squared will evaluate to $-1$, rather than Complex[-1,0]).
- Division will tend to produce more accurate results. The conventional division operator (/) is redefined to use quo, which doesn't round (quo is documented on page 661).
- Related to the previous point, rational numbers will be used in preference to floats when possible. Dividing one by two results in the rational number $\frac{1}{2}$, rather than $0.5$ (or $0$, the result of normal integer division).

See also: Matrix (page 769), Rational (page 796), Complex (page 737), Prime (page 791)

- Without mathn:

Download **samples/slmathn_1.rb**

```
require 'matrix'
36/16                       # =>   2
Math.sqrt(36/16)            # =>   1.4142135623731

Complex::I * Complex::I     # =>   (-1+0i)

(36/16)**-2                 # =>   1/4
(36.0/16.0)**-2             # =>   0.197530864197531
(-36/16)**-2                # =>   1/9

(36/16)**(1/2)              # =>   1
(-36/16)**(1/2)             # =>   1

(36/16)**(-1/2)             # =>   1/2
(-36/16)**(-1/2)            # =>   -1/3

Matrix.diagonal(6,7,8)/3    # =>
```

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

- With mathn:

Download **samples/slmathn_2.rb**

```
require 'mathn'
36/16                       # =>  9/4
Math.sqrt(36/16)            # =>  3/2

Complex::I * Complex::I     # =>  -1

(36/16)**-2                 # =>  16/81
(36.0/16.0)**-2             # =>  0.197530864197531
(-36/16)**-2                # =>  16/81

(36/16)**(1/2)              # =>  3/2
(-36/16)**(1/2)             # =>  (9.18485099360515e-17+1.5i)

(36/16)**(-1/2)             # =>  2/3
(-36/16)**(-1/2)            # =>  (4.08215599715784e-17-0.666666666666667i)

Matrix.diagonal(6,7,8)/3    # =>
```

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 7/3 & 0 \\ 0 & 0 & 8/3 \end{pmatrix}$$

Library

## Matrix — Matrix and Vector Manipulation

The matrix library defines classes Matrix and Vector, representing rectangular matrices and vectors. As well as the normal arithmetic operations, they provide methods for matrix-specific functions (such as rank, inverse, and determinants) and a number of constructor methods (for creating special-case matrices—zero, identity, diagonal, singular, and vector).

1.9

As of Ruby 1.9, matrices use quo internally for division, so rational numbers may be returned as a result of integer division. In prior versions of Ruby, you'd need to include the mathn library to achieve this.

Download **samples/slmatrix_1.rb**

```
require 'matrix'

m1 = Matrix[ [2, 1], [-1, 1] ]              # => ⎛ 2  1⎞
                                                ⎝-1  1⎠
m1[0,1]                                     # => 1
m1.inv                                      # => ⎛ 1/3  -1/3⎞
                                                ⎝ 1/3   2/3⎠
m1 * m1.inv                                 # => ⎛ 1/1  0/1⎞
                                                ⎝ 0/1  1/1⎠
m1.determinant                              # => 3/1
m1.singular?                                # => false
v1 = Vector[3, 4]                           # => Vector[3, 4]
v1.covector                                 # => ( 3  4 )
m1 * v1                                     # => Vector[10, 1]
m2 = Matrix[ [1,2,3], [4,5,6], [7,8,9] ]    # => ⎛1  2  3⎞
                                                ⎜4  5  6⎟
                                                ⎝7  8  9⎠
m2.minor(1, 2, 1, 2)                        # => ⎛5  6⎞
                                                ⎝8  9⎠
```

Library

## MiniTest — Unit Testing Framework

1.9

New in Ruby 1.9, MiniTest is now the standard unit testing framework supplied with Ruby. The MiniTest library contains classes for unit tests, mock objects, and a (trivial) subset of RSpec-style testing syntax.

The unit testing framework is similar to the original Test::Unit framework. However, if you want functionality that is the same as Test::Unit, use the Test::Unit wrappers for MiniTest—simply require "test/unit" as normal.

Chapter 13 on page 198 contains a tutorial on unit testing with Ruby.

Library

## Monitor — Monitor-Based Synchronization

Monitors are a mutual-exclusion mechanism. They allow separate threads to define shared resources that will be accessed exclusively, and they provide a mechanism for a thread to wait for resources to become available in a controlled way.

The monitor library actually defines three separate ways of using monitors: as a parent class, as a mixin, and as a extension to a particular object. In this section, we document the module form of Monitor. The class form is effectively identical. In both the class form and when including MonitorMixin in an existing class, it is essential to invoke super in the class's initialize method.

See also: Thread (page 705)

(The following example would be better written using fibers.)

Download samples/slmonitor_1.rb

```
require 'monitor'
require 'mathn'
numbers = []
numbers.extend(MonitorMixin)
number_added = numbers.new_cond
# Reporter thread
consumer = Thread.new do
  5.times do
    numbers.synchronize do
      number_added.wait_while { numbers.empty? }
      puts numbers.shift
    end
  end
end
# Prime number generator thread
generator = Thread.new do
  p = Prime.new
  5.times do
    numbers.synchronize do
      numbers << p.succ
      number_added.signal
    end
  end
end
generator.join
consumer.join
```

*produces:*

```
Prime::new is obsolete. use Prime::instance or class methods of Prime.
2
3
5
7
11
```

Library

## Mutex_m — Mutex Mix-In

mutex_m is a variant of class Mutex (documented on page 612) that allows mutex facilities to be mixed into any object.

The Mutex_m module defines methods that correspond to those in Mutex but with the prefix mu_ (so that lock is defined as mu_lock and so on). These are then aliased to the original Mutex names.

See also: Mutex (page 612), Thread (page 705)

Download **samples/slmutexm_1.rb**

```
require 'mutex_m'

class Counter
  include Mutex_m
  attr_reader :count
  def initialize
    @count = 0
    super
  end
  def tick
    lock
    @count += 1
    unlock
  end
end

c = Counter.new

t1 = Thread.new { 100_000.times {  c.tick } }
t2 = Thread.new { 100_000.times {  c.tick } }

t1.join
t2.join

c.count   # =>   200000
```

Library

## Net::FTP — FTP Client

The net/ftp library implements a File Transfer Protocol (FTP) client. As well as data transfer commands (getbinaryfile, gettextfile, list, putbinaryfile, and puttextfile), the library supports the full complement of server commands (acct, chdir, delete, mdtm, mkdir, nlst, rename, rmdir, pwd, size, status, and system). Anonymous and password-authenticated sessions are supported. Connections may be active or passive.

See also: open-uri (page 782)

Download **samples/slnetftp_1.rb**

```
require 'net/ftp'
ftp = Net::FTP.new('ftp.ruby-lang.org')
ftp.login
ftp.chdir('pub/ruby/doc')
puts ftp.list('*txt')
ftp.getbinaryfile('MD5SUM.txt', 'md5sum.txt', 1024)
ftp.close
puts File.read('md5sum.txt')
```

*produces:*

```
-rw-r--r-- 1 1027 100 3060 Jan 21 11:21 MD5SUM.txt
-rw-r--r-- 1 1027 100 3436 Jan 21 11:22 SHA1SUM.txt
d529768c828c930c49b3766d13dc1f2c ruby-man-1.4.6-jp.tar.gz
8eed63fec14a719df26247fb8384db5e ruby-man-1.4.6.tar.gz
623b5d889c1f15b8a50fe0b3b8ba4b0f ruby-man-ja-1.6.6-20011225-rd.tar.gz
5f37ef2d67ab1932881cd713989af6bf ruby-man-ja-html-20050214.tar.bz2
. . .
```

Library

## Net::HTTP — HTTP Client

The net/http library provides a simple client to fetch headers and web page contents using the HTTP protocol.

The get post, and head methods return a response object, with the content of the response accessible through the response's body method.

See also: OpenSSL (page 784), open-uri (page 782), URI (page 826)

- Opens a connection and fetch a page, displaying the response code and message, header information, and some of the body:

```
require 'net/http'
Net::HTTP.start('www.pragprog.com') do |http|
  response = http.get('/categories/new')
  puts "Code = #{response.code}"
  puts "Message = #{response.message}"
  response.each {|key, val| printf "%-14s = %-40.40s\n", key, val }
  p response.body[0, 55]
end
```

*produces:*

```
Code = 200
Message = OK
server         = nginx/0.6.34
date           = Mon, 13 Apr 2009 18:27:01 GMT
content-type   = text/html; charset=UTF-8
transfer-encoding = chunked
connection     = keep-alive
set-cookie     = _pragmatic_session_id=af478c8333e8e2966b
status         = 200 OK
x-runtime      = 84ms
etag           = "1506b34159b8e2c85ee772c25704ff60"
x-app-info     = master@9402370c1ccb75092090e9b7d014853ad
cache-control  = private, max-age=0, must-revalidate
"<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\"\n  \"ht"
```

- Fetches a single page, displaying the response code and message, header information, and some of the body:

```
require 'net/http'
response = Net::HTTP.get_response('www.pragprog.com',
                                  '/categories/new')
puts "Code = #{response.code}"
puts "Message = #{response.message}"
response.each {|key, val| printf "%-14s = %-40.40s\n", key, val }
p response.body[0, 55]
```

*produces:*

```
Code = 200
Message = OK
server         = nginx/0.6.34
```

```
date           = Mon, 13 Apr 2009 18:27:02 GMT
content-type   = text/html; charset=UTF-8
connection     = keep-alive
set-cookie     = _pragmatic_session_id=e987d6f51627348841
status         = 200 OK
x-runtime      = 77ms
etag           = "1506b34159b8e2c85ee772c25704ff60"
x-app-info     = master@9402370c1ccb75092090e9b7d014853ad
cache-control  = private, max-age=0, must-revalidate
content-length = 22986
"<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\"\n  \"ht"
```

- Follows redirections (the open-uri library does this automatically). This code comes from the RDoc documentation.

```
require 'net/http'
require 'uri'
def fetch(uri_str, limit=10)
  fail 'http redirect too deep' if limit.zero?
  puts "Trying: #{uri_str}"
  response = Net::HTTP.get_response(URI.parse(uri_str))
  case response
  when Net::HTTPSuccess     then response
  when Net::HTTPRedirection then fetch(response['location'], limit-1)
  else response.error!
  end
end
response = fetch('http://www.ruby-lang.org')
p response.body[0, 50]
```

*produces:*

```
Trying: http://www.ruby-lang.org
Trying: http://www.ruby-lang.org/en/
"<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\"\n"
```

- Searches our site for things about Ruby and lists the authors. (This would be tidier using Hpricot,[3] but this doesn't run on Ruby 1.9 as I write this.)

```
require 'net/http'
response = Net::HTTP.post_form(URI.parse('http://pragprog.com/search'),
                               "q" => "ruby")
puts response.body.scan(%r{<p class="by-line">by (.*?)</p>})[0,3]
```

*produces:*

```
Dave Thomas, with Chad Fowler and Andy Hunt
Bruce Tate
Lyle Johnson
```

3. http://code.whytheluckystiff.net/hpricot/

Library

## Net::IMAP — Access an IMAP Mail Server

The Internet Mail Access Protocol (IMAP) is used to allow mail clients to access mail servers. It supports plain-text login and the IMAP login and CRAM-MD5 authentication mechanisms. Once connected, the library supports threading, so multiple interactions with the server may take place at the same time.

The examples that follow are taken with minor modifications from the RDoc documentation in the library source file.

The TMail gem provides an interface for creating and parsing e-mail messages.

See also: Net::POP (page 777)

- Lists senders and subjects of messages to “dave” in the inbox:

Download **samples/slnetimap_1.rb**

```
require 'net/imap'
imap = Net::IMAP.new('my.mailserver.com')
imap.authenticate('LOGIN', 'dave', 'secret')
imap.examine('INBOX')
puts "Message count: #{ imap.responses["EXISTS"]}"
imap.search(["TO", "dave"]).each do |message_id|
  envelope = imap.fetch(message_id, "ENVELOPE")[0].attr["ENVELOPE"]
     puts "#{envelope.from[0].name}: \t#{envelope.subject}"
end
```

- Moves all messages with a date in April 2008 from the folder `Mail/sent-mail` to `Mail/sent-apr08`:

Download **samples/slnetimap_2.rb**

```
require 'net/imap'
imap = Net::IMAP.new('my.mailserver.com')
imap.authenticate('LOGIN', 'dave', 'secret')
imap.select('Mail/sent-mail')
if not imap.list('Mail/', 'sent-apr08')
  imap.create('Mail/sent-apr08')
end
imap.search(["BEFORE", "01-May-2008",
             "SINCE",  "1-Apr-2008"]).each do |message_id|
  imap.copy(message_id, "Mail/sent-apr08")
  imap.store(message_id, "+FLAGS", [:Deleted])
end
imap.expunge
```

Library

## Net::POP — Access a POP Mail Server

The net/pop library provides a simple client to fetch and delete mail on a Post Office Protocol (POP) server.

The class Net::POP3 is used to access a POP server, returning a list of Net::POPMail objects, one per message stored on the server. These POPMail objects are then used to fetch and/or delete individual messages. The TMail gem provides an interface for creating and parsing e-mail messages.

The library also provides class APOP, an alternative to the POP3 class that performs authentication.

Download **samples/slnetpop_1.rb**

```
require 'net/pop'
pop = Net::POP3.new('server.ruby-stuff.com')
pop.start('joe', 'secret') do |server|
  msg = server.mails[0]
  # Print the 'From:' header line
  from = msg.header.split("\r\n").grep(/^From: /)[0]
  puts from
  puts
  puts "Full message:"
  text = msg.pop
  puts text
end
```

Download **samples/slnetpop_2.rb**

*produces:*

```
From: dave@facet.ruby-stuff.com (Dave Thomas)

Full message:
Return-Path: <dave@facet.ruby-stuff.com>
Received: from facet.ruby-stuff.com (facet.ruby-stuff.com [10.96.0.122])
        by pragprog.com (8.11.6/8.11.6) with ESMTP id i2PJMW701809
        for <joe@carat.ruby-stuff.com>; Thu, 25 Mar 2008 13:22:32 -0600
Received: by facet.ruby-stuff.com (Postfix, from userid 502)
        id 4AF228B1BD; Thu, 25 Mar 2008 13:22:36 -0600 (CST)
To: joe@carat.ruby-stuff.com
Subject: Try out the new features!
Message-Id: <20080325192236.4AF228B1BD@facet.ruby-stuff.com>
Date: Thu, 25 Mar 2008 13:22:36 -0600 (CST)
From: dave@facet.ruby-stuff.com (Dave Thomas)
Status: RO

Ruby 1.9 has even more new features, both in
the core language and in the supplied libraries.

Try it out!
```

Library

## Net::SMTP — Simple SMTP Client

The net/smtp library provides a simple client to send electronic mail using the Simple Mail Transfer Protocol (SMTP). It does not assist in the creation of the message payload—it simply delivers messages once an RFC822 message has been constructed. The TMail gem provides an interface for creating and parsing e-mail messages.

- Sends an e-mail from a string:

  Download **samples/slnetsmtp_1.rb**

```
require 'net/smtp'
msg = "Subject: Test\n\nNow is the time\n"
Net::SMTP.start('pragprog.com') do |smtp|
  smtp.send_message(msg, 'dave@pragprog.com', ['dave'])
end
```

- Sends an e-mail using an SMTP object and an adapter:

  Download **samples/slnetsmtp_2.rb**

```
require 'net/smtp'
Net::SMTP::start('pragprog.com', 25, "pragprog.com") do |smtp|
  smtp.open_message_stream('dave@pragprog.com', # from
                           [ 'dave' ]          # to
                          ) do |stream|
    stream.puts "Subject: Test1"
    stream.puts
    stream.puts "And so is this"
  end
end
```

- Sends an e-mail to a server requiring CRAM-MD5 authentication:

  Download **samples/slnetsmtp_3.rb**

```
require 'net/smtp'
msg = "Subject: Test\n\nNow is the time\n"
Net::SMTP.start('pragprog.com', 25, 'pragprog.com',
                'user', 'password', :cram_md5) do |smtp|
  smtp.send_message(msg, 'dave@pragprog.com', ['dave'])
end
```

Library

## Net::Telnet — Telnet Client

The net/telnet library provides a complete implementation of a telnet client and includes features that make it a convenient mechanism for interacting with nontelnet services.

Class Net::Telnet delegates to class Socket. As a result, the methods of Socket and its parent, class IO, are available through Net::Telnet objects.

- Connects to a localhost, runs the date command, and disconnects:

  Download samples/slnettelnet_1.rb

```
require 'net/telnet'
tn = Net::Telnet.new({})
tn.login "guest", "secret"
tn.cmd "date"   # =>   "Mon Apr 13 13:27:04 CDT 2009\n"
tn.close
```

- The methods new, cmd, login, and waitfor take an optional block. If present, the block is passed output from the server as it is received by the routine. This can be used to provide real-time output, rather than waiting (for example) for a login to complete before displaying the server's response.

  Download samples/slnettelnet_2.rb

```
require 'net/telnet'
tn = Net::Telnet.new({})     {|str| print str }
tn.login("guest", "secret")  {|str| print str }
tn.cmd("date")               {|str| print str }
tn.close
```

  *produces:*

```
Connected to localhost.
Darwin/BSD (dave-2.home) (ttys012)
login: guest
Password:Last login: Thu Mar  5 13:23:25 from 0.0.0.0
$ date
Mon Apr 13 13:27:04 CDT 2009
$
```

- Gets the time from an NTP server:

  Download samples/slnettelnet_3.rb

```
require 'net/telnet'
tn = Net::Telnet.new('Host'       => 'time.nonexistent.org',
                     'Port'       => 'time',
                     'Timeout'    => 60,
                     'Telnetmode' => false)
atomic_time = tn.recv(4).unpack('N')[0]
puts "Atomic time: " + Time.at(atomic_time - 2208988800).to_s
puts "Local time:  " + Time.now.to_s
```

  *produces:*

```
Atomic time: 2009-04-13 13:27:04 -0500
Local time:  2009-04-13 13:27:07 -0500
```

Library

# NKF — Interface to Network Kanji Filter

1.9

The NKF module is a wrapper around Itaru Ichikawa's Network Kanji Filter (NKF) library (version 1.7). It provides functions to guess at the encoding of JIS, EUC, and SJIS streams and to convert from one encoding to another. Even though Ruby 1.9 now supports these encodings natively, this library is still useful for guessing encodings.

1.9

- As of Ruby 1.9, NFK uses the built-in encoding objects:

  Download **samples/slnkf_1.rb**

```
require 'nkf'
NKF::AUTO   # =>   nil
NKF::JIS    # =>   #<Encoding:ISO-2022-JP (dummy)>
NKF::EUC    # =>   #<Encoding:EUC-JP>
NKF::SJIS   # =>   #<Encoding:Shift_JIS>
```

- Guesses at the encoding of a string. (Thanks to Nobu Nakada for the examples on this page.)

  Download **samples/slnkf_2.rb**

```
require 'nkf'
p NKF.guess("Yukihiro Matsumoto")
p NKF.guess("\e$B$^$D$b$H$f$-$R$m\e(B")
p NKF.guess("\244\336\244\304\244\342\244\310\244\346\244\255\244\322\244\355")
p NKF.guess("\202\334\202\302\202\340\202\306\202\344\202\253\202\320\202\353")
```

  *produces:*

```
#<Encoding:US-ASCII>
#<Encoding:ISO-2022-JP (dummy)>
#<Encoding:EUC-JP>
#<Encoding:Shift_JIS>
```

- The NKF.nfk method takes two parameters. The first is a set of options, passed on to the NKF library. The second is the string to translate. The following examples assume that your console is set up to accommodate Japanese characters. The text at the end of the three ruby commands is Yukihiro Matsumoto.

```
$ ruby -e 'p *ARGV'  まつもと ゆきひろ
"\244\336\244\304\244\342\244\310\244\346\244\255\244\322\244\355"
$ ruby -rnkf -e 'p NKF.nkf(*ARGV)' - -Es  まつもと ゆきひろ
"\202\334\202\302\202\340\202\306\202\344\202\253\202\320\202\353"
$ ruby -rnkf -e 'p NKF.nkf(*ARGV)' - -Ej  まつもと ゆきひろ
"\e$B$^$D$b$H$f$-$R$m\e(B"
```

Library

## Observable — The Observer Pattern

The Observer pattern, also known as Publish/Subscribe, provides a simple mechanism for one object (the source) to inform a set of interested third-party objects when its state changes (see *Design Patterns* [GHJV95]). In the Ruby implementation, the notifying class mixes in the module Observable, which provides the methods for managing the associated observer objects. The observers must implement the update method to receive notifications.

Download **samples/slobserver_1.rb**

```
require 'observer'
class CheckWaterTemperature # Periodically check the water
  include Observable
  def run
    last_temp = nil
    loop do
      temp = Temperature.fetch   # external class...
      puts "Current temperature: #{temp}"
      if temp != last_temp
        changed                  # notify observers
        notify_observers(Time.now, temp)
        last_temp = temp
      end
    end
  end
end
class Warner
  def initialize(&limit)
    @limit = limit
  end
  def update(time, temp)         # callback for observer
    if @limit.call(temp)
      puts "--- #{time.to_s}: Temperature outside range: #{temp}"
    end
  end
end
checker = CheckWaterTemperature.new
checker.add_observer(Warner.new {|t| t < 80})
checker.add_observer(Warner.new {|t| t > 120})
checker.run
```

*produces:*

```
Current temperature: 83
Current temperature: 75
--- 2009-04-13 13:27:05 -0500: Temperature outside range: 75
Current temperature: 90
Current temperature: 134
--- 2009-04-13 13:27:05 -0500: Temperature outside range: 134
Current temperature: 134
Current temperature: 112
Current temperature: 79
--- 2009-04-13 13:27:05 -0500: Temperature outside range: 79
```

Library

## open-uri — Treat FTP and HTTP Resources as Files

The open-uri library extends Kernel#open, allowing it to accept URIs for FTP and HTTP as well as local filenames. Once opened, these resources can be treated as if they were local files, accessed using conventional IO methods. The URI passed to open is either a string containing an HTTP or FTP URL or a URI object (described on page 826). When opening an HTTP resource, the method automatically handles redirection and proxies. When using an FTP resource, the method logs in as an anonymous user.

The IO object returned by open in these cases is extended to support methods that return metainformation from the request: content_type, charset, content_encoding, last_modified, status, base_uri, meta.

See also: URI (page 826)

Download **samples/slopen-uri_1.rb**

```
require 'open-uri'
require 'pp'
open('http://ruby-lang.org') do |f|
  puts "URI: #{f.base_uri}"
  puts "Content-type: #{f.content_type}, charset: #{f.charset}"
  puts "Encoding: #{f.content_encoding}"
  puts "Last modified: #{f.last_modified}"
  puts "Status: #{f.status.inspect}"
  pp f.meta
  puts "----"
  3.times {|i| puts "#{i}: #{f.gets}" }
end
```

*produces:*

```
URI: http://www.ruby-lang.org/en/
Content-type: text/html, charset: utf-8
Encoding: []
Last modified:
Status: ["200", "OK"]
{"date"=>"Mon, 13 Apr 2009 18:27:07 GMT",
 "server"=>
  "Apache/2.2.3 (Debian) DAV/2 SVN/1.4.2 mod_ruby/1.2.6 Ruby/1.8.5(2006-08-25)
mod_ssl/2.2.3 OpenSSL/0.9.8c",
 "transfer-encoding"=>"chunked",
 "content-type"=>"text/html;charset=utf-8"}
----
0: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
1:    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
2: <html xmlns="http://www.w3.org/1999/xhtml">
```

Library

## Open3 — Run Subprocess and Connect to All Streams

Runs a command in a subprocess. Data written to *stdin* can be read by the subprocess, and data written to standard output and standard error in the subprocess will be available on the *stdout* and *stderr* streams. The subprocess is actually run as a grandchild, and as a result, Process#waitall cannot be used to wait for its termination (hence the sleep in the following example). Note also that you probably cannot assume that the application's output and error streams will not be buffered, so output may not arrive when you expect it to.

Download **samples/slopen3_1.rb**

```
require 'open3'
Open3.popen3('bc') do | stdin, stdout, stderr |
  Thread.new { loop { puts "STDOUT stream: #{stdout.gets}" } }
  Thread.new { loop { puts "STDERR stream: #{stderr.gets}" } }
  stdin.puts "3 * 4"
  stdin.puts "1 / 0"
  stdin.puts "2 ^ 5"
  sleep 0.1
end
```

*produces:*

```
STDOUT stream: 12
STDOUT stream: 32
STDERR stream: Runtime error (func=(main), adr=3): Divide by zero
```

Library

## OpenSSL — SSL Library

**Only if:** OpenSSL library available

The Ruby OpenSSL extension wraps the freely available OpenSSL library. It provides the Secure Sockets Layer and Transport Layer Security (SSL and TLS) protocols, allowing for secure communications over networks. The library provides functions for certificate creation and management, message signing, and encryption/decryption. It also provides wrappers to simplify access to `https` servers, along with secure FTP. The interface to the library is large (roughly 330 methods), but the average Ruby user will probably use only a small subset of the library's capabilities.

See also: Net::FTP (page 773), Net::HTTP (page 774), Socket (page 811)

- Accesses a secure website using HTTPS. Note that SSL is used to tunnel to the site, but the requested page also requires standard HTTP basic authorization.

  Download **samples/slopenssl_1.rb**

```
require 'net/https'
USER = "xxx"
PW   = "yyy"
site = Net::HTTP.new("www.securestuff.com", 443)
site.use_ssl = true
response = site.get2("/cgi-bin/cokerecipe.cgi",
                     'Authorization' => 'Basic ' +
                     ["#{USER}:#{PW}"].pack('m').strip)
```

- Creates a socket that uses SSL. This isn't a good example of accessing a website. However, it illustrates how a socket can be encrypted.

  Download **samples/slopenssl_2.rb**

```
require 'socket'
require 'openssl'
socket = TCPSocket.new("www.secure-stuff.com", 443)
ssl_context = OpenSSL::SSL::SSLContext.new()
unless ssl_context.verify_mode
  warn "warning: peer certificate won't be verified this session."
  ssl_context.verify_mode = OpenSSL::SSL::VERIFY_NONE
end
sslsocket = OpenSSL::SSL::SSLSocket.new(socket, ssl_context)
sslsocket.sync_close = true
sslsocket.connect
sslsocket.puts("GET /secret-info.shtml")
while line =  sslsocket.gets
  p line
end
```

Library

## OptionParser — Option Parsing

OptionParser is a flexible and extensible way to parse command-line arguments. It has a particularly rich abstraction of the concept of an option.

- An option can have multiple short names (options preceded by a single hyphen) and multiple long names (options preceded by two hyphens). Thus, an option that displays help may be available as -h, -?, --help, and --about. Users may abbreviate long option names to the shortest nonambiguous prefix.
- An option may be specified as having no argument, an optional argument, or a required argument. Arguments can be validated against patterns or lists of valid values.
- Arguments may be returned as objects of any type (not just strings). The argument type system is extensible (we add Date handling in the example).
- Arguments can have one or more lines of descriptive text, used when generating usage information.

Options are specified using the on and def methods. These methods take a variable number of arguments that cumulatively build a definition of each option. The arguments accepted by these methods are listed in Table 28.2 on the next page.

See also: GetoptLong (page 759)

Download samples/sloptparse_2.rb

```
require 'optparse'
require 'date'
# Add Dates as a new option type
OptionParser.accept(Date, /(\d+)-(\d+)-(\d+)/) do |d, mon, day, year|
  Date.new(year.to_i, mon.to_i, day.to_i)
end
opts = OptionParser.new
opts.on("-x")                          {|val| puts "-x seen" }
opts.on("-s", "--size VAL", Integer) {|val| puts "-s #{val}" }
opts.on("-a", "--at DATE",  Date)    {|val| puts "-a #{val}" }
my_argv = [ "--size", "1234", "-x", "-a", "12-25-2008", "fred", "wilma" ]
rest = opts.parse(*my_argv)
puts "Remainder = #{rest.join(', ')}"
puts opts.to_s
```

*produces:*

```
-s 1234
-x seen
-a 2008-12-25
Remainder = fred, wilma
Usage: myprog [options]
    -x
    -s, --size VAL
    -a, --at DATE
```

Table 28.2. Option Definition Arguments

**"-x" "-xARG" "-x=ARG" "-x[OPT]" "-x[=OPT]" "-x PLACE"**
: Option has short name x. First form has no argument, next two have mandatory argument, next two have optional argument, last specifies argument follows option. The short names may also be specified as a range (such as "-[a-c]").

**"--*switch*" "--*switch*=ARG" "--*switch*=[OPT]" "--*switch* PLACE"**
: Option has long name switch. First form has no argument, next has a mandatory argument, the next has an optional argument, and the last specifies the argument follows the switch.

**"--no-*switch*"**
: Defines a option whose default value is false.

**"=ARG" "=[OPT]"**
: Argument for this option is mandatory or optional. For example, the following code says there's an option known by the aliases -x, -y, and -z that takes a mandatory argument, shown in the usage as N:
```
opt.on("-x", "-y", "-z", "=N")
```

**"description"**
: Any string that doesn't start – or = is used as a description for this option in the summary. Multiple descriptions may be given; they'll be shown on additional lines.

**/pattern/**
: Any argument must match the given pattern.

**array**
: Argument must be one of the values from array.

**proc or method**
: Argument type conversion is performed by the given proc or method (rather than using the block associated with the on or def method call).

**ClassName**
: Argument must match that defined for ClassName, which may be predefined or added using OptionParser.accept. Built-in argument classes are

  **Object:** Any string. No conversion. This is the default.

  **String:** Any nonempty string. No conversion.

  **Integer:** Ruby/C-like integer with optional sign (0ddd is octal, 0bddd binary, 0xddd hexadecimal). Converts to Integer.

  **Float:** Float number format. Converts to Float.

  **Numeric:** Generic numeric format. Converts to Integer for integers, Float for floats.

  **Array:** Argument must be of list of strings separated by a comma.

  **OptionParser::DecimalInteger:** Decimal integer. Converted to Integer.

  **OptionParser::OctalInteger:** Ruby/C-like octal/hexadecimal/binary integer.

  **OptionParser::DecimalNumeric:** Decimal integer/float number. Integers converted to Integer, floats to Float.

  **TrueClass, FalseClass:** Boolean switch.

Library

## OpenStruct — Open (dynamic) Structure

An open structure is an object whose attributes are created dynamically when first assigned. In other words, if *obj* is an instance of an OpenStruct, then the statement obj.abc=1 will create the attribute *abc* in *obj* and then assign the value 1 to it.

Download **samples/slostruct_1.rb**

```
require 'ostruct'

os = OpenStruct.new( "f1" => "one", :f2 => "two" )
os.f3 = "cat"
os.f4 = 99
os.f1   # =>   "one"
os.f2   # =>   "two"
os.f3   # =>   "cat"
os.f4   # =>   99
```

Because OpenStruct uses method_missing and because it is a subclass of Object, you can't name field with the same names as Object's instance methods. In the following example, the access to ice.freeze is a call to Object#freeze:

Download **samples/slostruct_2.rb**

```
require 'ostruct'

ice = OpenStruct.new
ice.freeze = "yes"
ice.freeze   # =>   #<OpenStruct freeze="yes">
```

Library

## Pathname — Representation of File Paths

A Pathname represents the absolute or relative name of a file. It has two distinct uses. First, it allows manipulation of the parts of a file path (extracting components, building new paths, and so on). Second (and somewhat confusingly), it acts as a façade for some methods in classes Dir, File, and module FileTest, forwarding on calls for the file named by the Pathname object.

See also: File (page 506)

- Path name manipulation:

Download **samples/slpathname_1.rb**

```
require 'pathname'

p1 = Pathname.new("/usr/bin")
p2 = Pathname.new("ruby")
p3 = p1 + p2
p4 = p2 + p1
p3.parent          # =>   #<Pathname:/usr/bin>
p3.parent.parent   # =>   #<Pathname:/usr>
p1.absolute?       # =>   true
p2.absolute?       # =>   false
p3.split           # =>   [#<Pathname:/usr/bin>, #<Pathname:ruby>]

p5 = Pathname.new("testdir")
puts p5.realpath
puts p5.children
```

*produces:*

```
/Users/dave/BS2/titles/RUBY3/Book/testdir
testdir/config.h
testdir/main.rb
```

- Path name as proxy for file and directory status requests:

Download **samples/slpathname_3.rb**

```
require 'pathname'

p1 = Pathname.new("/usr/bin/ruby")
p1.file?                        # =>   true
p1.directory?                   # =>   false
p1.executable?                  # =>   true
p1.size                         # =>   38304

p2 = Pathname.new("testfile")   # =>   #<Pathname:testfile>

p2.read                         # =>   "This is line one\nThis is line
                                       two\nThis is line three\nAnd so
                                       on...\n"
p2.readlines                    # =>   ["This is line one\n", "This is
                                       line two\n", "This is line
                                       three\n", "And so on...\n"]
```

Library

# PP — Pretty-print Objects

PP uses the PrettyPrint library to format the results of inspecting Ruby objects. As well as the methods in the class, it defines a global function, pp, which works like the existing p method but formats its output.

PP has a default layout for all Ruby objects. However, you can override the way it handles a class by defining the method pretty_print, which takes a PP object as a parameter. It should use that PP object's methods text, breakable, nest, group, and pp to format its output (see PrettyPrint for details).

See also: JSON (page 765), PrettyPrint (page 790), YAML (page 831)

- Compares "p" and "pp":

Download **samples/slpp_1.rb**

```
require 'pp'
Customer = Struct.new(:name, :sex, :dob, :country)
cust = Customer.new("Walter Wall", "Male", "12/25/1960", "Niue")
puts "Regular print"
p cust
puts "\nPretty print"
pp cust
```

*produces:*

```
Regular print
#<struct Customer name="Walter Wall", sex="Male", dob="12/25/1960",
 country="Niue">

Pretty print
#<struct Customer
 name="Walter Wall",
 sex="Male",
 dob="12/25/1960",
 country="Niue">
```

- You can tell PP not to display an object if it has already displayed it:

Download **samples/slpp_2.rb**

```
require 'pp'
a = "string"
b = [ a ]
c = [ b, b ]
PP.sharing_detection = false
pp c
PP.sharing_detection = true
pp c
```

*produces:*

```
[["string"], ["string"]]
[["string"], [...]]
```

Library

## PrettyPrint — General Pretty Printer

PrettyPrint implements a pretty printer for structured text. It handles details of wrapping, grouping, and indentation. The PP library uses PrettyPrint to generate more legible dumps of Ruby objects.

See also: PP (page 789)

- The following program prints a chart of Ruby's classes, showing subclasses as a bracketed list following the parent. To save some space, we show just the classes in the Numeric branch of the tree.

Download **samples/slprettyprint_1.rb**

```
require 'prettyprint'
@children = Hash.new { |h,k| h[k] = Array.new }
ObjectSpace.each_object(Class) do |cls|
  @children[cls.superclass] << cls if cls <= Numeric
end
def print_children_of(printer, cls)
  printer.text(cls.name)
  kids = @children[cls].sort_by(&:name)
  unless kids.empty?
    printer.group(0, " [", "]") do
      printer.nest(3) do
        printer.breakable
        kids.each_with_index do |k, i|
          printer.breakable unless i.zero?
          print_children_of(printer, k)
        end
      end
      printer.breakable
    end
  end
end
printer = PrettyPrint.new($stdout, 30)
print_children_of(printer, Object)
printer.flush
```

*produces:*

```
Object [
   Numeric [
      Complex
      Float
      Integer [
         Bignum
         Fixnum
      ]
      Rational
   ]
]
```

Library

## prime — Prime Numbers

Provides facilities for generating prime numbers, as well as factoring numbers. Note that the Prime class is a singleton.

See also: mathn (page 767)

- The prime library extends the number classes to include new functionality and adds a new class Prime:

```
require 'prime'

# 60 = 2**2 * 3 * 5
60.prime?           # =>   false
60.prime_division   # =>   [[2, 2], [3, 1], [5, 1]]
```

- You can also use it to generate sequences of primes:

```
require 'prime'
Prime.each {|p| puts p; break if p > 20 }
```

*produces:*

```
2
3
5
7
11
13
17
19
23
```

Library

## Profile — Profile Execution of a Ruby Program

The profile library is a trivial wrapper around the Profiler module, making it easy to profile the execution of an entire program. Profiling can be enabled from the command line using the -rprofile option or from within a source program by requiring the profile module.

1.9

Unlike Ruby 1.8, Ruby 1.9 does not profile primitive methods such as Fixnum#== and iFixnum#+. This helps boost Ruby's performance.

See also: Benchmark (page 731), Profiler__ (page 793)

Download **samples/slprofile_1.rb**

```
require 'profile'
def ackerman(m, n)
  if m == 0 then  n+1
  elsif n == 0 and m > 0 then ackerman(m-1, 1)
  else ackerman(m-1, ackerman(m, n-1))
  end
end
ackerman(3, 3)
```

*produces:*

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.00     0.04     0.04     2432     0.02     0.41  Object#ackerman
  0.00     0.04     0.00        1     0.00     0.00  Kernel.puts
  0.00     0.04     0.00        1     0.00     0.00  IO#puts
  0.00     0.04     0.00        1     0.00     0.00  Module#method_added
  0.00     0.04     0.00        2     0.00     0.00  IO#write
  0.00     0.04     0.00        1     0.00    40.00  #toplevel
```

Library

## Profiler__ Control Execution Profiling

The Profiler__ module can be used to collect a summary of the number of calls to, and the time spent in, methods in a Ruby program. The output is sorted by the total time spent in each method. The profile library is a convenience wrapper that profiles an entire program.

See also: Benchmark (page 731), profile (page 792)

Download **samples/slprofiler_1.rb**

```
require 'profiler'
# Omit definition of connection and fetching methods
def calc_discount(qty, price)
  case qty
  when 0..10 then 0.0
  when 11..99 then price * 0.05
  else price * 0.1
  end
end
def calc_sales_totals(rows)
   total_qty =  total_price = total_disc = 0
   rows.each do |row|
     total_qty   += row.qty
     total_price += row.price
     total_disc  += calc_discount(row.qty, row.price)
   end
end
connect_to_database
rows = read_sales_data
Profiler__::start_profile
calc_sales_totals(rows)
Profiler__::stop_profile
Profiler__::print_profile(STDOUT)
```

*produces:*

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 50.00     0.04     0.04      648     0.06     0.06  Range#include?
 25.00     0.06     0.02      324     0.06     0.22  Object#calc_discount
 12.50     0.07     0.01      648     0.02     0.08  Range#===
 12.50     0.08     0.01        1    10.00    80.00  Array#each
  0.00     0.08     0.00      648     0.00     0.00  S#qty
  0.00     0.08     0.00      648     0.00     0.00  Float#<=>
  0.00     0.08     0.00      648     0.00     0.00  Fixnum#<=>
  0.00     0.08     0.00      648     0.00     0.00  S#price
  0.00     0.08     0.00        3     0.00     0.00  Fixnum#+
  0.00     0.08     0.00        1     0.00    80.00  Object#calc_sales_totals
  0.00     0.08     0.00        1     0.00    80.00  #toplevel
```

Library

## PStore — Persistent Object Storage

The PStore class provides transactional, file-based, persistent storage of Ruby objects. Each PStore can store several object hierarchies. Each hierarchy has a root, identified by a key (often a string). At the start of a PStore transaction, these hierarchies are read from a disk file and made available to the Ruby program. At the end of the transaction, the hierarchies are written back to the file. Any changes made to objects in these hierarchies are therefore saved on disk, to be read at the start of the next transaction that uses that file.

In normal use, a PStore object is created and then is used one or more times to control a transaction. Within the body of the transaction, any object hierarchies that had previously been saved are made available, and any changes to object hierarchies, and any new hierarchies, are written back to the file at the end.

- The following example stores two hierarchies in a PStore. The first, identified by the key "names", is an array of strings. The second, identified by "tree", is a simple binary tree.

Download **samples/slpstore_1.rb**

```
require 'pstore'
require 'pp'
class T
  def initialize(val, left=nil, right=nil)
    @val, @left, @right = val, left, right
  end
  def to_a
    [ @val, @left.to_a, @right.to_a ]
  end
end
store = PStore.new("/tmp/store")
store.transaction do
   store['names'] = [ 'Douglas', 'Barenberg', 'Meyer' ]
   store['tree']  = T.new('top',
                      T.new('A', T.new('B')),
                      T.new('C', T.new('D', nil, T.new('E'))))
end
# now read it back in
store.transaction do
   puts "Roots: #{store.roots.join(', ')}"
   puts store['names'].join(', ')
   pp store['tree'].to_a
end
```

*produces:*

```
Roots: names, tree
Douglas, Barenberg, Meyer
["top",
 ["A", ["B", [], []], []],
 ["C", ["D", [], ["E", [], []]], []]]
```

Library

## PTY — Pseudo-Terminal Interface: Interact with External Processes

**Only if:** Unix with pty support

Many Unix platforms support a *pseudo-terminal*—a device pair where one end emulates a process running on a conventional terminal, and the other end can read and write that terminal as if it were a user looking at a screen and typing on a keyboard.

The PTY library provides the method spawn, which starts the given command (by default a shell), connecting it to one end of a pseudo-terminal. It then returns the reader and writer streams connected to that terminal, allowing your process to interact with the running process.

Working with pseudo-terminals can be tricky. See IO#expect on page 752 for a convenience method that makes life easier. You might also want to track down Ara T. Howard's Session module for an even simpler approach to driving subprocesses.[4]

See also: expect (page 752)

- Runs irb in a subshell and asks it to convert the string "cat" to uppercase:

Download **samples/slpty_1.rb**

```
require 'pty'
require 'expect'
$expect_verbose = true
PTY.spawn("/usr/local/rubybook/bin/ruby /usr/local/rubybook/bin/irb") do |reader,
writer, pid|
  reader.expect(/irb.*:0> /)
  writer.puts "'cat'.upcase"
  reader.expect("=> ")
  answer = reader.gets
  puts "Answer = #{answer}"
end
```

*produces:*

```
irb(main):001:0> 'cat'.upcase
=> Answer = "CAT"
```

4. Currently found at http://www.codeforpeople.com/lib/ruby/session/.

Library

## Rational — Rational Numbers

1.9

The Rational class is now built in to Ruby. The vestigial Rational library simply defines a few aliases for backward compatibility. For the classes Fixnum and Bignum, the following aliases are defined:

**Floating-point division**
quof is an alias for fdiv.

**Rational division**
rdiv is an alias for quo.

**Exponentiation**
power! and rpower are aliases for **.

Library

## Readline — Interface to GNU Readline Library

**Only if:** GNU readline present

The Readline module allows programs to prompt for and receive lines of user input. The module allows lines to be edited during entry, and command history allows previous commands to be recalled and edited. The history can be searched, allowing the user to (for example) recall a previous command containing the text *ruby*. Command completion allows context-sensitive shortcuts: tokens can be expanded in the command line under control of the invoking application. In typical GNU fashion, the underlying readline library supports more options than any user could need and emulates both vi and emacs key bindings.

- This meaningless program implements a trivial interpreter that can increment and decrement a value. It uses the Abbrev module (described on page 729) to expand abbreviated commands when the Tab key is pressed.

Download **samples/slreadline_1.rb**

```
require 'readline'
include Readline
require 'abbrev'
COMMANDS = %w{ exit inc dec }
ABBREV = COMMANDS.abbrev
Readline.completion_proc = proc do |string|
  ABBREV[string]
end
value = 0
loop do
  cmd = readline("wibble [#{value}]: ", true)
  break if cmd.nil?
  case cmd.strip
  when "exit"
    break
  when "inc"
    value += 1
  when "dec"
    value -= 1
  else
    puts "Invalid command #{cmd}"
  end
end
```

```
% ruby code/readline.rb
wibble [0]: inc
wibble [1]: <up-arrow>    => inc
wibble [2]: d<tab>        => dec
wibble [1]: ^r i          => inc
wibble [2]: exit
%
```

Library

## Resolv — DNS Client Library

The resolv library is a pure-Ruby implementation of a DNS client—it can be used to convert domain names into corresponding IP addresses. It also supports reverse lookups and the resolution of names in the local hosts file.

The resolv library exists to overcome a problem with the interaction of the standard operating system DNS lookup and the Ruby threading mechanism. On most operating systems, name resolution is synchronous: you issue the call to look up a name, and the call returns when an address has been fetched. Because this lookup often involves network traffic and because DNS servers can be slow, this call may take a (relatively) long time. During this time, the thread that issued the call is effectively suspended. Because Ruby does not use operating system threads, this means that the interpreter is effectively suspended while a DNS request is being executed from any running Ruby thread. This is sometimes unacceptable. Enter the resolv library. Because it is written in Ruby, it automatically participates in Ruby threading, and hence other Ruby threads can run while a DNS lookup is in progress in one thread.

Loading the additional library resolv-replace insinuates the resolv library into Ruby's socket library (see page 811).

- Uses the standard socket library to look up a name. A counter running in a separate thread is suspended while this takes place.

  Download **samples/slresolve_1.rb**

```
require 'socket'

count = 0
thread = Thread.new { Thread.pass; loop {   count += 1;  } }
IPSocket.getaddress("www.ruby-lang.org")   # =>   "221.186.184.68"
count                                      # =>   0
```

- Repeats the experiment but uses the resolv library to allow Ruby's threading to work in parallel:

  Download **samples/slresolve_2.rb**

```
require 'socket'
require 'resolv-replace'

count = 0
thread = Thread.new { Thread.pass; loop {   count += 1;  } }
IPSocket.getaddress("www.ruby-lang.org")   # =>   "221.186.184.68"
count                                      # =>   1334853
```

Library

## REXML — XML Processing Library

REXML is a pure-Ruby XML processing library, including DTD-compliant document parsing, XPath querying, and document generation. It supports both tree-based and stream-based document processing. Because it is written in Ruby, it is available on all platforms supporting Ruby. REXML has a full and complex interface—this section contains a few small examples.

- Assume the file demo.xml contains this:

```
<classes language="ruby">
  <class name="Numeric">
    Numeric represents all numbers.
    <class name="Float">
      Floating point numbers have a fraction and a mantissa.
    </class>
    <class name="Integer">
      Integers contain exact integral values.
      <class name="Fixnum">
        Fixnums are stored as machine ints.
      </class>
      <class name="Bignum">
        Bignums store arbitraty-sized integers.
      </class>
    </class>
  </class>
</classes>
```

- Reads and processes the XML:

Download **samples/slrexml_2.rb**

```
require 'rexml/document'
xml = REXML::Document.new(File.open("demo.xml"))
puts "Root element: #{xml.root.name}"
puts "\nThe names of all classes"
xml.elements.each("//class") {|c| puts c.attributes["name"] }
puts "\nThe description of Fixnum"
p xml.elements["//class[@name='Fixnum']"].text
```

*produces:*

```
Root element: classes

The names of all classes
Numeric
Float
Integer
Fixnum
Bignum

The description of Fixnum
"\n        Fixnums are stored as machine ints.\n      "
```

- Reads in a document, adds and deletes elements, and manipulates attributes before writing it back out:

Download **samples/slrexml_3.rb**

```
require 'rexml/document'
include REXML
xml = Document.new(File.open("demo.xml"))
cls = Element.new("class")
cls.attributes["name"] = "Rational"
cls.text = "Represents complex numbers"
# Remove Integer's children, and add our new node as
# the one after Integer
int = xml.elements["//class[@name='Integer']"]
int.delete_at(1)
int.delete_at(2)
int.next_sibling = cls
# Change all the 'name' attributes to class_name
xml.elements.each("//class") do |c|
  c.attributes['class_name'] = c.attributes['name']
  c.attributes.delete('name')
end
# and write it out with a XML declaration at the front
xml << XMLDecl.new
xml.write(STDOUT, 0)
```

*produces:*

```
<?xml version='1.0'?>
<classes language='ruby'>
<class class_name='Numeric'>
 Numeric represents all numbers.
<class class_name='Float'>
 Floating point numbers have a fraction and a mantissa.
</class>
<class class_name='Integer'>
 Integers contain exact integral values.
</class>
<class class_name='Rational'>
Represents complex numbers
</class>
</class>
</classes>
```

Library

## Rinda — Tuplespace Implementation

Tuplespaces are a distributed blackboard system. Processes may add tuples to the blackboard, and other processes may remove tuples from the blackboard that match a certain pattern. Originally presented by David Gelernter, tuplespaces offer an interesting scheme for distributed cooperation among heterogeneous processes.

Rinda, the Ruby implementation of tuplespaces, offers some interesting additions to the concept. In particular, the Rinda implementation uses the === operator to match tuples. This means that tuples may be matched using regular expressions, the classes of their elements, and the element values.

See also: DRb (page 747)

- The blackboard is a DRb server that offers a shared tuplespace:

Download **samples/slrinda_1.rb**

```
require 'rinda/tuplespace'
MY_URI = "druby://127.0.0.1:12131"
DRb.start_service(MY_URI, Rinda::TupleSpace.new)
DRb.thread.join
```

- The arithmetic agent accepts messages containing an arithmetic operator and two numbers. It stores the result back on the blackboard.

Download **samples/slrinda_2.rb**

```
require 'rinda/rinda'
MY_URI = "druby://127.0.0.1:12131"
DRb.start_service
ts = Rinda::TupleSpaceProxy.new(DRbObject.new(nil, MY_URI))
loop do
  op, v1, v2 = ts.take([ %r{^[-+/*]$}, Numeric, Numeric])
  ts.write(["result", v1.send(op, v2)])
end
```

- The client places tuples on the blackboard and reads back the result of each:

Download **samples/slrinda_3.rb**

```
require 'rinda/rinda'
MY_URI = "druby://127.0.0.1:12131"
DRb.start_service
ts = Rinda::TupleSpaceProxy.new(DRbObject.new(nil, MY_URI))
queries = [[ "+", 1, 2 ],  [ "*", 3, 4 ],  [ "/", 8, 2 ]]
queries.each do |q|
  ts.write(q)
  ans = ts.take(["result", nil])
  puts "#{q[1]} #{q[0]} #{q[2]} = #{ans[1]}"
end
```

*produces:*

```
1 + 2 = 3
3 * 4 = 12
8 / 2 = 4
```

Library

## Ripper — Parse Ruby Source

The ripper library gives you access to Ruby's parser. It can tokenize input, return lexical tokens, and return a nested S-expression. It also supports event-based parsing.

- Tokenize a line of Ruby code:

Download samples/slripper_1.rb

```
require "ripper"
content = "a=1;b=2;puts a+b"
Ripper.tokenize(content)   # =>   ["a", "=", "1", ";", "b", "=", "2",
                                  ";", "puts", " ", "a", "+", "b"]
```

- Does a lexical analysis, returning token types, values, line and column numbers:

Download samples/slripper_2.rb

```
require "ripper"
require "pp"
content = "a=1;b=2;puts a+b"
pp Ripper.lex(content)[0,5]
```

*produces:*

```
[[[1, 0], :on_ident, "a"],
 [[1, 1], :on_op, "="],
 [[1, 2], :on_int, "1"],
 [[1, 3], :on_semicolon, ";"],
 [[1, 4], :on_ident, "b"]]
```

- Returns the sexp representing a chunk of code:

Download samples/slripper_3.rb

```
require "ripper"
require "pp"
content = "a=1;b=2;puts a+b"
pp Ripper.sexp(content)
```

*produces:*

```
[:program,
 [[:assign, [:var_field, [:@ident, "a", [1, 0]]], [:@int, "1", [1, 2]]],
  [:assign, [:var_field, [:@ident, "b", [1, 4]]], [:@int, "2", [1, 6]]],
  [:command,
   [:@ident, "puts", [1, 8]],
   [:args_add_block,
    [[:binary,
      [:var_ref, [:@ident, "a", [1, 13]]],
      :+,
      [:var_ref, [:@ident, "b", [1, 15]]]]],
    false]]]]
```

- As a (silly) example of event-based lexical analysis, here's a program that finds class definitions and their associated comment blocks. For each, it outputs the class name and the comment. It might be considered the zeroth iteration of an RDoc-like program.

The parameter to parse is an accumulator—it is passed between event handlers and can be used to construct the result.

Download **samples/slripper_4.rb**

```
require 'ripper'
# This class handles parser events, extracting
# comments and attaching them to class definitions
class BabyRDoc < Ripper::Filter
  def initialize(*)
    super
    reset_state
  end
  def on_default(event, token, output)
    reset_state
    output
  end
  def on_sp(token, output) output end
  alias on_nil on_sp
  def on_comment(comment, output)
    @comment << comment.sub(/^\s*#\s*/, "    ")
    output
  end
  def on_kw(name, output)
    @expecting_class_name = (name == 'class')
    output
  end
  def on_const(name, output)
    if @expecting_class_name
      output << "#{name}:\n"
      output <<  @comment
    end
    reset_state
    output
  end
  private
  def reset_state
    @comment = ""
    @expecting_class_name = false
  end
end
BabyRDoc.new(File.read(__FILE__)).parse(STDOUT)
```

*produces:*

```
BabyRDoc:
    This class handles parser events, extracting
    comments and attaching them to class definitions
```

Library

## RSS — RSS Feed Generation and Parsing

Rich (or RDF) Site Summary, Really Simple Syndication—take your pick. RSS is the protocol of choice for disseminating news on the Internet. The Ruby RSS library supports creating and parsing streams compliant with RSS 0.9, RSS 1.0, and RSS 2.0.

- Reads and summarizes the latest stories from http://ruby-lang.org:

  Download **samples/slrss_1.rb**

```
require 'rss/2.0'
require 'open-uri'
open('http://ruby-lang.org/en/feeds/news.rss') do |http|
  response = http.read
  result = RSS::Parser.parse(response, false)
  puts "Channel: " + result.channel.title
  result.items.each_with_index do |item, i|
    puts "#{i+1}. #{item.title}" if i < 3
  end
end
```

  *produces:*

```
Channel: Ruby News
1. MountainWest RubyConf Schedule
2. Ruby 1.9.1 released
3. Server maintenance
```

- Generates some RSS information:

  Download **samples/slrss_2.rb**

```
require 'rss/0.9'
rss  = RSS::Rss.new("0.9")
chan = RSS::Rss::Channel.new
chan.title       = "The Daily Dave"
chan.description = "Dave's Feed"
chan.language    = "en-US"
chan.link        = "http://pragdave.pragprog.com"
rss.channel      = chan
image = RSS::Rss::Channel::Image.new
image.url   = "http://pragprog.com/pragdave.gif"
image.title = "PragDave"
image.link  = chan.link
chan.image  = image
3.times do |i|
  item = RSS::Rss::Channel::Item.new
  item.title       = "My News Number #{i}"
  item.link        = "http://pragprog.com/pragdave/story_#{i}"
  item.description = "This is a story about number #{i}"
  chan.items << item
end
puts rss.to_s
```

Library

## Scanf — Input Format Conversion

Implements a version of the C library scanf function, which extracts values from a string under the control of a format specifier.

The Ruby version of the library adds a scanf method to both class IO and class String. The version in IO applies the format string to the next line read from the receiver. The version in String applies the format string to the receiver. The library also adds the global method Kernel.scanf, which uses as its source the next line of standard input.

Scanf has one main advantage over using regular expressions to break apart a string: a regular expression extracts strings whereas scanf will return objects converted to the correct type.

- Splits a date string into its constituents:

Download **samples/slscanf_1.rb**

```
require 'scanf'

date = "2004-12-15"
year, month, day = date.scanf("%4d-%2d-%2d")
year         # =>   2004
month        # =>   12
day          # =>   15
year.class   # =>   Fixnum
```

- The block form of scanf applies the format multiple times to the input string, returning each set of results to the block:

Download **samples/slscanf_2.rb**

```
require 'scanf'
data = "cat:7 dog:9 cow:17 walrus:31"
data.scanf("%[^:]:%d ") do |animal, value|
  puts "A #{animal.strip} has #{value*1.4}"
end
```

*produces:*

```
A cat has 9.8
A dog has 12.6
A cow has 23.8
A walrus has 43.4
```

- Extracts hex numbers:

Download **samples/slscanf_3.rb**

```
require 'scanf'

data = "decaf bad"
data.scanf("%3x%2x%x")   # =>   [3564, 175, 2989]
```

Library

## SDBM — Interface to SDBM Database

The SDBM database implements a simple key/value persistence mechanism. Because the underlying SDBM library itself is provided with Ruby, there are no external dependencies, and SDBM should be available on all platforms supported by Ruby. SDBM database keys and values must be strings. SDBM databases are effectively hashlike.

See also: DBM (page 743), GDBM (page 758)

- Stores a record in a new database and then fetches it back. Unlike the DBM library, all values to SDBM must be strings (or implement to_str).

Download **samples/slsdbm_1.rb**

```
require 'sdbm'
require 'date'
SDBM.open("data.dbm") do |dbm|
  dbm['name'] = "Walter Wombat"
  dbm['dob']  = Date.new(1997, 12,25).to_s
  dbm['uses'] = "Ruby"
end
SDBM.open("data.dbm", nil) do |dbm|
  p dbm.keys
  p dbm['dob']
  p dbm['dob'].class
end
```

*produces:*

```
["name", "dob", "uses"]
"1997-12-25"
String
```

Library

## SecureRandom — Access to Secure Random Number Generators

Provides access to one of your operating system's secure random number generators. If the OpenSSL library is installed, the module uses it's random_bytes method. Otherwise, the module looks for and uses /dev/urandom or the CryptGenRandom method in the Windows API.

- Generates some random numbers:

Download **samples/slsecurerandom_1.rb**

```
require 'securerandom'
# Random floats such that 0.0 <= rand < 1.0
SecureRandom.random_number(0)      # =>   0.936650421906334
SecureRandom.random_number(0)      # =>   0.320008123694954

# Random integers such that 0 <= rand < 1000
SecureRandom.random_number(1000)   # =>   670
SecureRandom.random_number(1000)   # =>   486
```

- Generates 10 random bytes, returning the result as a hex string, a Base64 string, and a string of binary data. A different random string is returned for each call.

Download **samples/slsecurerandom_2.rb**

```
require 'securerandom'
SecureRandom.hex(10)            # =>   "e04f6894931d226d30a3"
SecureRandom.base64(10)         # =>   "BENl6hFc8BuklQ=="
SecureRandom.random_bytes(10)   # =>   "NqP\xB7d[\xD7\xF5k5"
```

Library

## Set — Implement Various Forms of Set

A Set is a collection of unique values (where uniqueness is determined using eql? and hash). Convenience methods let you build sets from enumerable objects.

- Basic set operations:

```
require 'set'

set1 = Set.new([:bear, :cat, :deer])

set1.include?(:bat)   # =>   false
set1.add(:fox)        # =>   #<Set: {:bear, :cat, :deer, :fox}>

partition = set1.classify {|element| element.to_s.length }

partition     # =>   {4=>#<Set: {:bear, :deer}>, 3=>#<Set: {:cat,
                    :fox}>}

set2 = [ :cat, :dog, :cow ].to_set
set1 | set2   # =>   #<Set: {:bear, :cat, :deer, :fox, :dog, :cow}>
set1 & set2   # =>   #<Set: {:cat}>
set1 - set2   # =>   #<Set: {:bear, :deer, :fox}>
set1 ^ set2   # =>   #<Set: {:dog, :cow, :bear, :deer, :fox}>
```

- Partitions the users in our /etc/passwd file into subsets where members of each subset have adjacent user IDs:

```
require 'etc'
require 'set'
users = []
Etc.passwd {|u| users << u }
related_users = users.to_set.divide do |u1, u2|
  (u1.uid - u2.uid).abs <= 1
end
related_users.each do |relatives|
  relatives.each {|u| print "#{u.uid}/#{u.name} " }
  puts
end
```

*produces:*

```
67/_ard
93/_calendar 92/_securityagent 91/_tokend
59/_devdocs 60/_sandbox 58/_serialnumberd
26/_lp 27/_postfix
54/_mcxalr 55/_pcastagent 56/_pcastserver
65/_mdnsresponder
4/_uucp
1/daemon 0/root
501/dave 502/juliet 503/testuser
```

Library

## Shellwords — Manipulate Shell Lines Using POSIX Semantics

Given a string representative of a shell command line, splits it into word tokens according to POSIX semantics. Also allows you to create properly escaped shell lines from individual words.

- Spaces between double or single quotes are treated as part of a word.
- Double quotes may be escaped using a backslash.
- Spaces escaped by a backslash are not used to separate words.
- Otherwise, tokens separated by whitespace are treated as words.

Download **samples/slshellwords_1.rb**

```
require 'shellwords'
include Shellwords

line = %{Code Ruby Be Happy!}
shellwords(line)                        # =>   ["Code", "Ruby", "Be",
                                               "Happy!"]

line = %{"Code Ruby" 'Be Happy'!}
shellwords(line)                        # =>   ["Code Ruby", "Be Happy!"]

line = %q{Code\ Ruby "Be Happy"!}
shellwords(line)                        # =>   ["Code Ruby", "Be Happy!"]

shelljoin(["Code Ruby", "Be Happy"])    # =>   Code\ Ruby Be\ Happy
```

In addition, the library adds shellsplit and shelljoin methods to classes String and CArray, respectively:

Download **samples/slshellwords_2.rb**

```
require 'shellwords'
include Shellwords

%{Code\\ Ruby Be Happy!}.shellsplit   # =>   ["Code Ruby", "Be", "Happy!"]
["Code Ruby", "Be Happy"].shelljoin   # =>   "Code\\ Ruby Be\\ Happy"
```

Library

## Singleton — The Singleton Pattern

The Singleton design pattern ensures that only one instance of a particular class may be created for the lifetime of a program (see *Design Patterns* [GHJV95]).

The singleton library makes this simple to implement. Mix the Singleton module into each class that is to be a singleton, and that class's new method will be made private. In its place, users of the class call the method instance, which returns a singleton instance of that class.

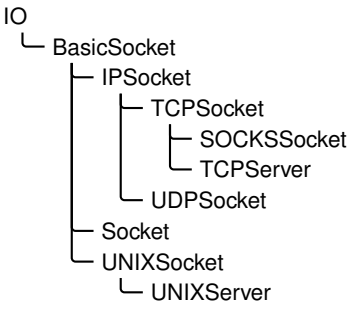
In this example, the two instances of MyClass are the same object:

Download **samples/slsingleton_1.rb**

```
require 'singleton'

class MyClass

  attr_accessor :data
  include Singleton
end

a = MyClass.instance   # =>   #<MyClass:0x128d4c>
b = MyClass.instance   # =>   #<MyClass:0x128d4c>

a.data = 123           # =>   123
b.data                 # =>   123
```

Library

## Socket — IP, TCP, Unix, and SOCKS Socket Access




The socket extension defines nine classes for accessing the socket-level communications of the underlying system. All of these classes are (indirect) subclasses of class IO, meaning that IO's methods can be used with socket connections.

The hierarchy of socket classes reflects the reality of network programming and hence is somewhat confusing. The BasicSocket class largely contains methods common to data transfer for all socket-based connections. It is subclassed to provide protocol-specific implementations: IPSocket, UNIXSocket (for domain sockets), and (indirectly) TCPSocket, UDPSocket, and SOCKSSocket.

BasicSocket is also subclassed by class Socket, which is a more generic interface to socket-oriented networking. Although classes such as TCPSocket are specific to a protocol, Socket objects can, with some work, be used regardless of protocol.

TCPSocket, SOCKSSocket, and UNIXSocket are each connection oriented. Each has a corresponding *xxxx*Server class, which implements the server end of a connection.

The socket libraries are something that you may never use directly. However, if you do use them, you'll need to know the details. For that reason, we've included a reference section covering the socket library methods in Appendix A on page 878.

The following code shows a trivial UDP server and client. For more examples see Appendix A:

Download samples/slsocket_1.rb

```
# Simple logger prints messages
# received on UDP port 12121
require 'socket'
socket = UDPSocket.new
socket.bind("127.0.0.1", 12121)
loop do
  msg, sender  = socket.recvfrom(100)
  host = sender[3]
  puts "#{Time.now}: #{host} '#{msg}'"
  STDOUT.flush
end
```

Download samples/slsocket_2.rb

```
# Exercise the logger
require 'socket'
log = UDPSocket.new
log.connect("127.0.0.1", 12121)
log.print "Up and Running!"
# process ... process ..
log.print "Done!"
```

*produces:*

```
2009-04-13 13:27:15 -0500: 127.0.0.1 'Up and Running!'
2009-04-13 13:27:15 -0500: 127.0.0.1 'Done!'
```

Library

## StringIO — Treat Strings as IO Objects

1.9

In some ways the distinction between strings and file contents is artificial: the contents of a file is basically a string that happens to live on disk, not in memory. The StringIO library aims to unify the two concepts, making strings act as if they were opened IO objects. Once a string is wrapped in a StringIO object, it can be read from and written to as if it were an open file. This can make unit testing a lot easier. It also lets you pass strings into classes and methods that were originally written to work with files. StringIO objects take their encoding from the string you pass in or the default external encoding is no string is passed.

- Reads and writes from a string:

Download **samples/slstringio_1.rb**

```
require 'stringio'

sio = StringIO.new("time flies like an arrow")
sio.read(5)             # =>   "time "
sio.read(5)             # =>   "flies"
sio.pos = 19
sio.read(5)             # =>   "arrow"
sio.rewind              # =>   0
sio.write("fruit")      # =>   5
sio.pos = 16
sio.write("a banana")   # =>   8
sio.rewind              # =>   0
sio.read                # =>   "fruitflies like a banana"
```

- Uses StringIO as a testing aid:

Download **samples/slstringio_2.rb**

```
require 'stringio'
require 'csv'
require 'test/unit'
class TestCSV < Test::Unit::TestCase
  def test_simple
    StringIO.open do |op|
      CSV(op) do |csv|
        csv << [ 1, "line 1", 27 ]
        csv << [ 2, nil, 123 ]
      end
      assert_equal("1,line 1,27\n2,,123\n", op.string)
    end
  end
end
```

*produces:*

```
Loaded suite /tmp/prog
Started
.
Finished in 0.013298 seconds.

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

Library

## StringScanner — Basic String Tokenizer

StringScanner objects progress through a string, matching (and optionally returning) tokens that match a given pattern. Unlike the built-in scan methods, StringScanner objects maintain a current position pointer in the string being examined, so each call resumes from the position in the string where the previous call left off. Pattern matches are anchored to this previous point.

- Implements a simple language:

Download **samples/slstrscan_1.rb**

```
require 'strscan'
# Handle the language:
#   set <var> = <value>
#   get <var>
values = {}
while line = gets
  scanner = StringScanner.new(line.chomp)
  scanner.scan(/(get|set)\s+/) or fail "Missing command"
  cmd = scanner[1]
  var_name = scanner.scan(/\w+/) or fail "Missing variable"
  case cmd
  when "get"
    puts "#{var_name} => #{values[var_name].inspect}"
  when "set"
    scanner.skip(/\s+=\s+/) or fail "Missing '='"
    value = scanner.rest
    values[var_name] = value
  else
    fail cmd
  end
end
```

*produces:*

```
% ruby code/strscan.rb
set a = dave
set b = hello
get b
b => "hello"
get a
a => "dave"
```

Library

## Syslog — Interface to Unix System Logging

**Only if:** Unix system with syslog

The Syslog class is a simple wrapper around the Unix syslog(3) library. It allows messages to be written at various severity levels to the logging daemon, where they are disseminated according to the configuration in syslog.conf. The following examples assume the log file is `/var/log/system.log`.

- Adds to our local system log. We'll log all the levels configured for the user facility for our system (which is every level except debug and info messages).

Download **samples/slsyslog_1.rb**

```
require 'syslog'
log = Syslog.open("test")   # "test" is the app name
log.debug("Warm and fuzzy greetings from your program")
log.info("Program starting")
log.notice("I said 'Hello!'")
log.warning("If you don't respond soon, I'm quitting")
log.err("You haven't responded after %d milliseconds", 7)
log.alert("I'm telling your mother...")
log.emerg("I'm feeling totally crushed")
log.crit("Aarrgh....")
system("tail -6 /var/log/system.log")
```

*produces:*

```
Apr 13 13:27:15 dave-2 test[86448]: I said 'Hello!'
Apr 13 13:27:15 dave-2 test[86448]: If you don't respond soon, I'm quitting
Apr 13 13:27:15 dave-2 test[86448]: You haven't responded after 7 milliseconds
Apr 13 13:27:15 dave-2 test[86448]: I'm telling your mother...
Apr 13 13:27:15 dave-2 test[86448]: I'm feeling totally crushed
Apr 13 13:27:15 dave-2 test[86448]: Aarrgh....
```

- Logs only errors and above:

Download **samples/slsyslog_3.rb**

```
require 'syslog'
log = Syslog.open("test")
log.mask = Syslog::LOG_UPTO(Syslog::LOG_ERR)
log.debug("Warm and fuzzy greetings from your program")
log.info("Program starting")
log.notice("I said 'Hello!'")
log.warning("If you don't respond soon, I'm quitting")
log.err("You haven't responded after %d milliseconds", 7)
log.alert("I'm telling your mother...")
log.emerg("I'm feeling totally crushed")
log.crit("Aarrgh....")
system("tail -4 /var/log/system.log")
```

*produces:*

```
Apr 13 13:27:16 dave-2 test[86454]: You haven't responded after 7 milliseconds
Apr 13 13:27:16 dave-2 test[86454]: I'm telling your mother...
Apr 13 13:27:16 dave-2 test[86454]: I'm feeling totally crushed
Apr 13 13:27:16 dave-2 test[86454]: Aarrgh....
```

Library

## Tempfile — Temporary File Support

Class Tempfile creates managed temporary files. Although they behave the same as any other IO objects, temporary files are automatically deleted when the Ruby program terminates. Once a Tempfile object has been created, the underlying file may be opened and closed a number of times in succession.

Tempfile does not directly inherit from IO. Instead, it delegates calls to a File object. From the programmer's perspective, apart from the unusual new, open, and close semantics, a Tempfile object behaves as if it were an IO object.

If you don't specify a directory to hold temporary files when you create them, the tmpdir library will be used to find a system-dependent location.

See also: tmpdir (page 822)

Download **samples/sltempfile_1.rb**

```
require 'tempfile'
tf = Tempfile.new("afile")
tf.path   # =>   "/var/folders/.../-Tmp-/afile20090413-86464-iwdzcf-0"
tf.puts("Cosi Fan Tutte")
tf.close
tf.open
tf.gets   # =>   "Cosi Fan Tutte\n"
tf.close(true)
```

Library

## Test::Unit — Unit Testing Framework

Test::Unit is a unit testing framework based on the original SUnit Smalltalk framework. It provides a structure in which unit tests may be organized, selected, and run. Tests can be run from the command line or using one of several GUI-based interfaces.

Chapter 13 on page 198 contains a tutorial on Test::Unit.

We could have a simple playlist class, designed to store and retrieve songs:

Download **samples/sltestunit_1.rb**

```
require 'code/testunit/song.rb'
require 'forwardable'
class Playlist
  extend Forwardable
  def_delegator(:@list, :<<, :add_song)
  def_delegators(:@list, :size, :empty?)
  def initialize
    @list = []
  end
  def find(title)
    @list.find {|song| song.title == title}
  end
end
```

We can write unit tests to exercise this class. The Test::Unit framework is smart enough to run the tests in a test class if no main program is supplied.

Download **samples/sltestunit_2.rb**

```
require 'test/unit'
require 'code/testunit/playlist.rb'
class TestPlaylist < Test::Unit::TestCase
  def test_adding
    pl = Playlist.new
    assert_empty(pl)
    assert_nil(pl.find("My Way"))
    pl.add_song(Song.new("My Way", "Sinatra"))
    assert_equal(1, pl.size)
    s = pl.find("My Way")
    refute_nil(s)
    assert_equal("Sinatra", s.artist)
    assert_nil(pl.find("Chicago"))
    # .. and so on
  end
end
```

*produces:*

```
Loaded suite /tmp/prog
Started
.
Finished in 0.002641 seconds.

1 tests, 7 assertions, 0 failures, 0 errors, 0 skips
```

Library

## thread — Utility Functionality for Threading

The thread library adds some utility functions and classes for supporting threads. Much of this has been superseded by the Monitor class, but the thread library contains two classes, Queue and SizedQueue, that are still useful. Both classes implement a thread-safe queue that can be used to pass objects between producers and consumers in multiple threads. The Queue object implements a unbounded queue. A SizedQueue is told its capacity; any producer that tries to add an object when the queue is at that capacity will block until a consumer has removed an object.

- The following example was provided by Robert Kellner. It has three consumers taking objects from an unsized queue. Those objects are provided by two producers, which each add three items.

Download **samples/slthread_1.rb**

```
require 'thread'
queue = Queue.new

consumers = (1..3).map do |i|
  Thread.new("consumer #{i}") do |name|
    begin
      obj = queue.deq
      print "#{name}: consumed #{obj.inspect}\n"
    end until obj == :END_OF_WORK
  end
end

producers = (1..2).map do |i|
  Thread.new("producer #{i}") do |name|
    3.times do |j|
      queue.enq("Item #{j} from #{name}")
    end
  end
end

producers.each(&:join)
consumers.size.times { queue.enq(:END_OF_WORK) }
consumers.each(&:join)
```

*produces:*

```
consumer 1: consumed "Item 0 from producer 2"
consumer 1: consumed "Item 0 from producer 1"
consumer 2: consumed "Item 1 from producer 2"
consumer 1: consumed "Item 1 from producer 1"
consumer 2: consumed "Item 2 from producer 2"
consumer 3: consumed "Item 2 from producer 1"
consumer 2: consumed :END_OF_WORK
consumer 3: consumed :END_OF_WORK
consumer 1: consumed :END_OF_WORK
```

Library

## ThreadsWait — Wait for Multiple Threads to Terminate

Class ThreadsWait handles the termination of a group of thread objects. It provides methods to allow you to check for termination of any managed thread and to wait for all managed threads to terminate.

The following example kicks off a number of threads that each wait for a slightly shorter length of time before terminating and returning their thread number. Using ThreadsWait, we can capture these threads as they terminate, either individually or as a group.

Download **samples/slthwait_1.rb**

```
require 'thwait'

group = ThreadsWait.new

# construct threads that wait for 1 second, .9 second, etc.
# add each to the group

9.times do |i|
  thread = Thread.new(i) {|index| sleep 1.0 - index/10.0; index }
  group.join_nowait(thread)
end

# any threads finished?
group.finished?              # =>   false

# wait for one to finish
group.next_wait.value        # =>   8

# wait for 5 more to finish
5.times { group.next_wait }  # =>   5

# wait for next one to finish
group.next_wait.value        # =>   2

# and then wait for all the rest
group.all_waits              # =>   nil
```

Library

## Time — Extended Functionality for Class Time

The time library adds functionality to the built-in class Time, supporting date and/or time formats used by RFC 2822 (e-mail), RFC 2616 (HTTP), and ISO 8601 (the subset used by XML schema).

```
require 'time'
Time.rfc2822("Thu, 1 Apr 2008 16:32:45 CST")
                                    → 2008-04-01 17:32:45 -0500

Time.rfc2822("Thu, 1 Apr 2008 16:32:45 -0600")
                                    → 2008-04-01 17:32:45 -0500

Time.now.rfc2822                    → Mon, 13 Apr 2009 13:27:18 -0500

Time.httpdate("Thu, 01 Apr 2008 16:32:45 GMT")
                                    → 2008-04-01 11:32:45 -0500

Time.httpdate("Thursday, 01-Apr-04 16:32:45 GMT")
                                    → 2004-04-01 16:32:45 UTC

Time.httpdate("Thu Apr 1 16:32:45 2008")
                                    → 2008-04-01 16:32:45 UTC

Time.now.httpdate                   → Mon, 13 Apr 2009 18:27:18 GMT

Time.xmlschema("2008-04-01T16:32:45")
                                    → 2008-04-01 16:32:45 -0500

Time.xmlschema("2008-04-01T16:32:45.12-06:00")
                                    → 2008-04-01 22:32:45 UTC

Time.now.xmlschema                  → 2009-04-13T13:27:18-05:00
```

Library

## Timeout — Run a Block with Timeout

The Timeout.timeout method takes a parameter representing a timeout period in seconds, an optional exception parameter, and a block. The block is executed, and a timer is run concurrently. If the block terminates before the timeout, timeout returns the value of the block. Otherwise, the exception (default Timeout::Error) is raised.

Download **samples/sltimeout_1.rb**

```
require 'timeout'
for snooze in 1..2
  puts "About to sleep for #{snooze}"
  begin
    Timeout::timeout(1.5) do |timeout_length|
      puts "Timeout period is #{timeout_length}"
      sleep(snooze)
      puts "That was refreshing"
    end
  rescue Timeout::Error
    puts "Woken up early!!"
  end
end
```

*produces:*

```
About to sleep for 1
Timeout period is 1.5
That was refreshing
About to sleep for 2
Timeout period is 1.5
Woken up early!!
```

Be careful when using timeouts—you may find them interrupting system calls that you cannot reliably restart, resulting in possible data loss.

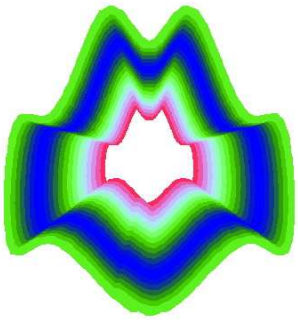
Library

## Tk — Wrapper for Tcl/Tk

**Only if:** Tk library installed

Of all the Ruby options for creating GUIs, the Tk library is probably the most widely supported, running on Windows, Linux, Mac OS X, and other Unix-like platforms.[5] Although it doesn't produce the prettiest interfaces, Tk is functional and relatively simple to program.

Download **samples/sltk_1.rb**



```
require 'tk'
include Math
TkRoot.new do |root|
  title "Curves"
  geometry "400x400"
 TkCanvas.new(root) do |canvas|
  width 400
  height 400
  pack('side'=>'top', 'fill'=>'both', 'expand'=>'yes')
    points = [ ]
    10.upto(30) do |scale|
      (0.0).step(2*PI,0.1) do |i|
        new_x = 5*scale*sin(i) + 200  +  scale*sin(i*2)
        new_y = 5*scale*cos(i) + 200  + scale*cos(i*6)
        points << [ new_x, new_y ]
        f = scale/5.0
        r = (Math.sin(f)+1)*127.0
        g = (Math.cos(2*f)+1)*127.0
        b = (Math.sin(3*f)+1)*127.0
        col = sprintf("#%02x%02x%02x", r.to_i, g.to_i, b.to_i)
        if points.size == 3
          TkcLine.new(canvas,
                      points[0][0], points[0][1],
                      points[1][0], points[1][1],
                      points[2][0], points[2][1],
                      'smooth'=>'on',
                      'width'=> 7,
                      'fill'     => col,
                      'capstyle' => 'round')
          points.shift
        end
      end
    end
  end
end
Tk.mainloop
```

5. All these environments require that the Tcl/Tk libraries are installed before the Ruby Tk extension can be used.

Library

## tmpdir — System-Independent Temporary Directory Location

The tmpdir library adds the tmpdir method to class Dir. This method returns the path to a temporary directory that *should* be writable by the current process. (This will not be true if none of the well-known temporary directories is writable and if the current working directory is also not writable.) Candidate directories include those referenced by the environment variables TMPDIR, TMP, TEMP, and USERPROFILE, the directory /tmp, and (on Windows boxes) the temp subdirectory of the Windows or System directory.

Download **samples/sltmpdir_1.rb**

```
require 'tmpdir'

Dir.tmpdir   # =>   "/var/folders/a4/a4-daQQOG4anplm9DAY+TE+++TI/-Tmp-"

ENV['TMPDIR'] = "/wibble"   # doesn't exist
ENV['TMP']    = "/sbin"     # not writable
ENV['TEMP']   = "/Users/dave/tmp" # just right

Dir.tmpdir   # =>   "/Users/dave/tmp"
```

The mktmpdir method can be used to create a new temporary directory:

Download **samples/sltmpdir_2.rb**

```
require 'tmpdir'
name = Dir.mktmpdir
# .. process, process, process ..
Dir.rmdir(name)
```

Library

## Tracer — Trace Program Execution

The tracer library uses Kernel.set_trace_func to trace all or part of a Ruby program's execution. The traced lines show the thread number, file, line number, class, event, and source line. The events shown are - for a change of line, > for a call, < for a return, C for a class definition, and E for the end of a definition.

- You can trace an entire program by including the tracer library from the command line:

```
class Account
  def initialize(balance)
    @balance = balance
  end
  def debit(amt)
    if @balance < amt
      fail "Insufficient funds"
    else
      @balance -= amt
    end
  end
end
acct = Account.new(100)
acct.debit(40)
```

```
% ruby -r tracer account.rb
#0:account.rb:1::-: class Account
#0:account.rb:1:Class:>: class Account
#0:account.rb:1:Class:<: class Account
#0:account.rb:1::C: class Account
#0:account.rb:2::-:   def initialize(balance)
#0:account.rb:2:Module:>:   def initialize(balance)
#0:account.rb:2:Module:<:   def initialize(balance)
#0:account.rb:5::-:   def debit(amt)
#0:account.rb:5:Module:>:   def debit(amt)
#0:account.rb:5:Module:<:   def debit(amt)
#0:account.rb:1::E: class Account
#0:account.rb:13::-: acct = Account.new(100)
#0:account.rb:13:Class:>: acct = Account.new(100)
#0:account.rb:2:Account:>:   def initialize(balance)
#0:account.rb:3:Account:-:     @balance = balance
#0:account.rb:13:Account:<: acct = Account.new(100)
#0:account.rb:13:Class:<: acct = Account.new(100)
#0:account.rb:14::-: acct.debit(40)
#0:account.rb:5:Account:>:   def debit(amt)
#0:account.rb:6:Account:-:     if @balance < amt
#0:account.rb:6:Account:-:     if @balance < amt
#0:account.rb:6:Fixnum:>:     if @balance < amt
#0:account.rb:6:Fixnum:<:     if @balance < amt
#0:account.rb:9:Account:-:       @balance -= amt
#0:account.rb:9:Fixnum:>:       @balance -= amt
#0:account.rb:9:Fixnum:<:       @balance -= amt
#0:account.rb:9:Account:<:       @balance -= amt
```

- You can also use tracer objects to trace just a portion of your code and use filters to select what to trace:

```
require 'tracer'

class Account
  def initialize(balance)
    @balance = balance
  end
  def debit(amt)
    if @balance < amt
      fail "Insufficient funds"
    else
      @balance -= amt
    end
  end
end

tracer = Tracer.new
tracer.add_filter lambda {|event, *rest| event == "line" }
acct = Account.new(100)
tracer.on do
  acct.debit(40)
end
```

```
#0:account.rb:20::-:   acct.debit(40)
#0:account.rb:8:Account:-:     if @balance < amt
#0:account.rb:8:Account:-:     if @balance < amt
#0:account.rb:11:Account:-:       @balance -= amt
```

Library

## TSort — Topological Sort

Given a set of dependencies between nodes (where each node depends on zero or more other nodes and there are no cycles in the graph of dependencies), a topological sort will return a list of the nodes ordered such that no node follows a node that depends on it. One use for this is scheduling tasks, where the order means that you will complete the dependencies before you start any task that depends on them. The make program uses a topological sort to order its execution.

In this library's implementation, you mix in the TSort module and define two methods: tsort_each_node, which yields each node in turn, and tsort_each_child, which, given a node, yields each of that nodes dependencies.

- Given the set of dependencies among the steps for making a piña colada, what is the optimum order for undertaking the steps?

Download **samples/sltsort_1.rb**

```
require 'tsort'
class Tasks
  include TSort
  def initialize
    @dependencies = {}
  end
  def add_dependency(task, *relies_on)
    @dependencies[task] = relies_on
  end
  def tsort_each_node(&block)
    @dependencies.each_key(&block)
  end
  def tsort_each_child(node, &block)
    deps = @dependencies[node]
    deps.each(&block) if deps
  end
end
tasks = Tasks.new
tasks.add_dependency(:add_rum,       :open_blender)
tasks.add_dependency(:add_pc_mix,    :open_blender)
tasks.add_dependency(:add_ice,       :open_blender)
tasks.add_dependency(:close_blender, :add_rum, :add_pc_mix, :add_ice)
tasks.add_dependency(:blend_mix,     :close_blender)
tasks.add_dependency(:pour_drink,    :blend_mix)
tasks.add_dependency(:pour_drink,    :open_blender)
puts tasks.tsort
```

*produces:*

```
open_blender
add_rum
add_pc_mix
add_ice
close_blender
blend_mix
pour_drink
```

Library

## un — Command-Line Interface to FileUtils

Why un? When you invoke it from the command line with the -r option to Ruby, it spells -run. This pun gives a hint as to the intent of the library: it lets you run commands (in this case, a subset of the methods in FileUtils) from the command line. In theory this gives you an operating system–independent set of file manipulation commands, possibly useful when writing portable Makefiles.

See also: FileUtils (page 755)

- The available commands are as follows:

```
% ruby -run -e cp -- <options> source dest
% ruby -run -e ln -- <options> target linkname
% ruby -run -e mv -- <options> source dest
% ruby -run -e rm -- <options> file
% ruby -run -e mkdir -- <options> dirs
% ruby -run -e rmdir -- <options> dirs
% ruby -run -e install -- <options> source dest
% ruby -run -e chmod -- <options> octal_mode file
% ruby -run -e touch -- <options> file
```

Note the use of -- to tell the Ruby interpreter that options to the program follow.

You can get a list of all available commands with this:

Download **samples/slun_1.rb**

```
% ruby -run -e help
```

For help on a particular command, append the command's name:

Download **samples/slun_2.rb**

```
% ruby -run -e help mkdir
```

Library

# URI — RFC 2396 Uniform Resource Identifier (URI) Support

URI encapsulates the concept of a Uniform Resource Identifier (URI), a way of specifying some kind of (potentially networked) resource. URIs are a superset of URLs: URLs (such as the addresses of web pages) allow specification of addresses by location, and URIs also allow specification by name.

URIs consist of a scheme (such as http, mailto, ftp, and so on), followed by structured data identifying the resource within the scheme.

URI has factory methods that take a URI string and return a subclass of URI specific to the scheme. The library explicitly supports the ftp, http, https, ldap, and mailto schemes; others will be treated as generic URIs. The module also has convenience methods to escape and unescape URIs. The class Net::HTTP accepts URI objects where a URL parameter is expected.

See also: open-uri (page 782), Net::HTTP (page 774)

Download **samples/sluri_1.rb**

```
require 'uri'

uri = URI.parse("http://pragprog.com:1234/mypage.cgi?q=ruby")
uri.class      # =>   URI::HTTP
uri.scheme     # =>   "http"
uri.host       # =>   "pragprog.com"
uri.port       # =>   1234
uri.path       # =>   "/mypage.cgi"
uri.query      # =>   "q=ruby"

uri = URI.parse("mailto:ruby@pragprog.com?Subject=help&body=info")
uri.class      # =>   URI::MailTo
uri.scheme     # =>   "mailto"
uri.to         # =>   "ruby@pragprog.com"
uri.headers    # =>   [["Subject", "help"], ["body", "info"]]

uri = URI.parse("ftp://dave@anon.com:/pub/ruby;type=i")
uri.class      # =>   URI::FTP
uri.scheme     # =>   "ftp"
uri.host       # =>   "anon.com"
uri.port       # =>   21
uri.path       # =>   "pub/ruby"
uri.typecode   # =>   "i"
```

Library

## WeakRef — Support for Weak References

In Ruby, objects are not eligible for garbage collection if references still exist to them. Normally, this is a Good Thing—it would be disconcerting to have an object simply evaporate while you were using it. However, sometimes you may need more flexibility. For example, you might want to implement an in-memory cache of commonly used file contents. As you read more files, the cache grows. At some point, you may run low on memory. The garbage collector will be invoked, but the objects in the cache are all referenced by the cache data structures and so will not be deleted.

A weak reference behaves like any normal object reference with one important exception—the referenced object may be garbage collected, even while references to it exist. In the cache example, if the cached files were accessed using weak references, once memory runs low, they will be garbage collected, freeing memory for the rest of the application.

- Weak references introduce a slight complexity. Because the object referenced can be deleted by garbage collection at any time, code that accesses these objects must take care to ensure that the references are valid. Two techniques can be used. First, the code can reference the objects normally. Any attempt to reference an object that has been garbage collected will raise a WeakRef::RefError exception.

  Download **samples/slweakref_1.rb**

  ```
  require 'weakref'
  # Generate lots of small strings. Hopefully the early ones will have
  # been garbage collected...
  refs = (1..10000).map {|i| WeakRef.new("#{i}") }
  puts "Last element is #{refs.last}"
  puts "First element is #{refs.first}"
  ```

  *produces:*

  ```
  Last element is 10000
  prog.rb:6:in `<main>': Invalid Reference - probably recycled
   (WeakRef::RefError)
  ```

- Alternatively, use the WeakRef#weakref_alive? method to check that a reference is valid before using it. Garbage collection must be disabled during the test and subsequent reference to the object. In a single-threaded program, you could use something like this:

  Download **samples/slweakref_2.rb**

  ```
  ref = WeakRef.new(some_object)
  # .. some time later
  gc_was_disabled = GC.disable
  if ref.weakref_alive?
    # do stuff with 'ref'
  end
  GC.enable unless gc_was_disabled
  ```

Library

## WEBrick — Web Server Toolkit

WEBrick is a pure-Ruby framework for implementing HTTP-based servers. The standard library includes WEBrick services that implement a standard web server (serving files and directory listings) and servlets supporting CGI, erb, file download, and the mounting of Ruby lambdas.

More examples of WEBrick start on page 314.

- The following code mounts two Ruby procs on a web server. Requests to the URI http://localhost:2000/hello run one proc, and requests to http://localhost:2000/bye run the other.

Download samples/slwebrick_1.rb

```
#!/usr/bin/ruby
require 'webrick'
include WEBrick
hello_proc = lambda do |req, resp|
  resp['Content-Type'] = "text/html"
  resp.body = %{
      <html><body>
        Hello. You're calling from a #{req['User-Agent']}
       <p>
        I see parameters: #{req.query.keys.join(', ')}
      </body></html>
  }
end
bye_proc = lambda do |req, resp|
  resp['Content-Type'] = "text/html"
  resp.body = %{
      <html><body>
        <h3>Goodbye!</h3>
      </body></html>
  }
end

hello =  HTTPServlet::ProcHandler.new(hello_proc)
bye   =  HTTPServlet::ProcHandler.new(bye_proc)
s = HTTPServer.new(:Port => 2000)
s.mount("/hello", hello)
s.mount("/bye",   bye)
trap("INT"){ s.shutdown }
s.start
```

Library

## WIN32OLE — Windows Automation

**Only if:** Windows

Interface to Windows automation, allowing Ruby code to interact with Windows applications. The Ruby interface to Windows is discussed in more detail in Chapter 21 on page 316.

- Opens Internet Explorer and asks it to display our home page:

Download **samples/slwin32ole_1.rb**

```
ie = WIN32OLE.new('InternetExplorer.Application')
ie.visible = true
ie.navigate("http://www.pragprog.com")
```

- Creates a new chart in Microsoft Excel and then rotates it:

Download **samples/slwin32ole_2.rb**

```
require 'win32ole'
#   -4100 is the value for the Excel constant xl3DColumn.
ChartTypeVal = -4100;
excel = WIN32OLE.new("excel.application")
# Create and rotate the chart
excel['Visible'] = TRUE
excel.Workbooks.Add()
excel.Range("a1")['Value'] = 3
excel.Range("a2")['Value'] = 2
excel.Range("a3")['Value'] = 1
excel.Range("a1:a3").Select()
excelchart = excel.Charts.Add()
excelchart['Type'] = ChartTypeVal
30.step(180, 5) do |rot|
  excelchart.rotation = rot
  sleep(0.1)
end
excel.ActiveWorkbook.Close(0)
excel.Quit()
```

Library

## XMLRPC — Remote Procedure Calls using XML-RPC

XMLRPC allows clients to invoke methods on networked servers using the XML-RPC protocol. Communications take place over HTTP. The server may run in the context of a web server, in which case ports 80 or 443 (for SSL) will typically be used. The server may also be run stand-alone. The Ruby XML-RPC server implementation supports operation as a CGI script, as a mod_ruby script, as a WEBrick handler, and as a stand-alone server. Basic authentication is supported, and clients can communicate with servers via proxies. Servers may throw FaultException errors—these generate the corresponding exception on the client (or optionally may be flagged as a status return to the call).

See also: dRuby (page 747), WEBrick (page 828)

- The following simple server accepts a temperature in Celsius and converts it to Fahrenheit. It runs within the context of the WEBrick web server.

Download **samples/slxmlrpc_1.rb**

```
require 'webrick'
require 'xmlrpc/server'
xml_servlet = XMLRPC::WEBrickServlet.new
xml_servlet.add_handler("convert_celcius") do |celcius|
  celcius*1.8 + 32
end
xml_servlet.add_multicall # Add support for multicall
server = WEBrick::HTTPServer.new(:Port => 2000)
server.mount("/RPC2", xml_servlet)
trap("INT"){ server.shutdown }
server.start
```

- This client makes calls to the temperature conversion server. Note that in the output we show both the server's logging and the client program's output.

Download **samples/slxmlrpc_2.rb**

```
require 'xmlrpc/client'
server = XMLRPC::Client.new("localhost", "/RPC2", 2000)
puts server.call("convert_celcius", 0)
puts server.call("convert_celcius", 100)
puts server.multicall(['convert_celcius', -10],
                      ['convert_celcius', 200])
```

*Produces:*

```
localhost - - [10/Apr/2008:17:17:23 CDT] "POST /RPC2 HTTP/1.1" 200 124 - -> /RPC2
32.0
localhost - - [10/Apr/2008:17:17:23 CDT] "POST /RPC2 HTTP/1.1" 200 125 - -> /RPC2
212.0
localhost - - [10/Apr/2008:17:17:23 CDT] "POST /RPC2 HTTP/1.1" 200 290 - -> /RPC2
14.0
392.0
```

Library

## YAML — Object Serialization/Deserialization

The YAML library (also described in the tutorial starting on page 433) serializes and deserializes Ruby object trees to and from an external, readable, plain-text format. YAML can be used as a portable object marshaling scheme, allowing objects to be passed in plain text between separate Ruby processes. In some cases, objects may also be exchanged between Ruby programs and programs in other languages that also have YAML support.

See also: json (page 765)

- YAML can be used to store an object tree in a flat file:

Download samples/slyaml_1.rb

```
require 'yaml'
tree = { :name => 'ruby',
         :uses => [ 'scripting', 'web', 'testing', 'etc' ]
       }
File.open("tree.yaml", "w") {|f| YAML.dump(tree, f)}
```

- Once stored, it can be read by another program:

Download samples/slyaml_2.rb

```
require 'yaml'
tree = YAML.load_file("tree.yaml")
tree[:uses][1]   # =>   "web"
```

- The YAML format is also a convenient way to store configuration information for programs. Because it is readable, it can be maintained by hand using a normal editor and then read as objects by programs. For example, a configuration file may contain the following:

Download samples/slyaml_3.rb

```
---
username: dave
prefs:
  background: dark
  foreground: cyan
  timeout: 30
```

We can use this in a program:

Download samples/slyaml_4.rb

```
require 'yaml'

config = YAML.load_file("code/config.yaml")
config["username"]                # =>   "dave"
config["prefs"]["timeout"] * 10   # =>   300
```

Library

## Zlib — Read and Write Compressed Files

**Only if:** zlib library available

The Zlib module is home to a number of classes for compressing and decompressing streams and for working with gzip-format compressed files. They also calculate zip checksums.

- Compresses /etc/passwd as a gzip file and then reads the result back:

Download **samples/slzlib_1.rb**

```
require 'zlib'
# These methods can take a filename
Zlib::GzipWriter.open("passwd.gz") do |gz|
  gz.write(File.read("/etc/passwd"))
end
system("ls -l /etc/passwd passwd.gz")
# or a stream
File.open("passwd.gz") do |f|
  gzip = Zlib::GzipReader.new(f)
  data = gzip.read.split(/\n/)
  puts data[15,3]
end
```

*produces:*

```
-rw-r--r--  1 root  wheel  2888 Sep 23  2007 /etc/passwd
-rw-rw-r--  1 dave  dave   1057 Apr 13 13:27 passwd.gz

daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
_lp:*:26:26:Printing Services:/var/spool/cups:/usr/bin/false
```

- Compresses data sent between two processes:

Download **samples/slzlib_2.rb**

```
require 'zlib'
rd, wr = IO.pipe
if fork
  rd.close
  zipper = Zlib::Deflate.new
  zipper << "This is a string "
  data = zipper.deflate("to compress", Zlib::FINISH)
  wr.write(data)
  wr.close
  Process.wait
else
  wr.close
  unzipper = Zlib::Inflate.new
  unzipper << rd.read
  puts "We got: #{unzipper.inflate(nil)}"
end
```

*produces:*

```
We got: This is a string to compress
```