

1 Ruby Tools

```
"goi/tools.rb" 1a ≡
  # Common Utilities
  <License 1b>
  <Required Modules 1c, ... >
  <definitions 2d, ... >
  <classes ?>
  module Goi
    <Goi classes 5b>
    module_function
    <Goi functions 2a, ... >
  end
  ◇

<License 1b> ≡
  # Copyright (C) 2003–2005, Javier Goizueta <javier@goizueta.info>
  #
  # This program is free software; you can redistribute it and/or
  # modify it under the terms of the GNU General Public License
  # as published by the Free Software Foundation; either version 2
  # of the License, or (at your option) any later version.
  ◇
```

Macro referenced in [1ad](#), [12](#).

```
<Required Modules 1c> ≡
  require 'rubygems'
  require 'fileutils'
  ◇
```

Macro defined by [1c](#), [9c](#), [10b](#).

Macro referenced in [1a](#).

2 Tests

```
"tools_test.rb" 1d ≡

  <License 1b>
  require 'test/unit/testcase'
  require 'test/unit/ui/console/testrunner'
  require 'test/unit/testsuite'
  require './goi/tools'
  include Goi
  <Tests definitions 2b, ... >
  class Test_tools < Test::Unit::TestCase
    <Tests 2c, ... >
  end
  Test::Unit::UI::Console::TestRunner.run(Test_tools)
  ◇
```

3 Optional Values

This is handy to extract parameters, e.g. from a hash. Note that having something like

```
return default_v unless v;v
# or
return v || default_v
```

Would not return the correct value if `v` is `false`.

```

<Goi functions 2a> ≡
  # Handle optional value (e.g. extract parameters from a hash)
  def opt(v, default_v)
    v.nil? ? default_v : v
  end
  ◇

```

Macro defined by [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#).

Macro referenced in [1a](#).

3.1 Tests

```

<Tests definitions 2b> ≡
  def oV_f(params={})
    a = opt(params[:a], 'A')
    b = opt(params[:b], true)
    c = opt(params[:c], 110)
    [a,b,c]
  end
  ◇

```

Macro defined by [2b](#), [6a](#).

Macro referenced in [1d](#).

```

<Tests 2c> ≡
  def test_opt
    assert_equal(['A', false, 110], oV_f(:b=>false))
    assert_equal(['B', true, 111], oV_f(:a=>'B', :c=>111))
  end
  ◇

```

Macro defined by [2c](#), [4bc](#), [5a](#), [6b](#).

Macro referenced in [1d](#).

4 Fold list

This operation is known under a variety of names, such as *fold* (Mathematica, etc.), *inject* (SmallTalk), *insertion* (functional programming), *stream* (RPL), *collect* (?). I'll use the name *accumulate* to avoid conflicts.

This function performs an operation combining all elements of a list; it accepts a block to perform the operation and an optional initial value, which is combined with the rest and is returned when the list is empty (it is usually the neuter element of the operation); if no initial value is passed, *nil* is used as the value for an empty list.

Note: Ruby 1.8.0 has a new built-in method in Array, *inject*, that acts just like the *accumulate* defined here.

```

<definitions 2d> ≡
  module Enumerable
    def accumulate(v0=nil)
      each do |v|
        if v0==nil
          v0=v
        else
          v0 = yield(v0,v)
        end
      end
      v0
    end
  end
  ◇

```

Macro defined by [2d](#), [4a](#), [6c](#).

Macro referenced in [1a](#).

The Ruby built-in library includes the *map/collect* (apply-to-all, *map*, *maplist* or *mapcar*). Here we'll add a

function to operate in parallel (element-wise) in several lists (such as DOLIST from RPL or MapThread from Mathematica)

```
< Go! functions 3 > ≡
  def multi_map(*arrays,&blk)

    # check number of arrays vs block arity (or block arity is 2)
    # check array sizes ...

    n = arrays.size
    return nil if n==0

    result = []

    # to make this valid for any Enumerable type ...
    arrays.map! { |arr| arr.to_a }

    n = arrays[0].size
    (0... n).each do |i|
      args = []
      arrays.each do |a|
        args << a[i]
      end
      if blk.arity==arrays.size
        result << yield(*args)
      else
        # assert blk.arity==2
        result << args.accumulate(&blk)
      end
    end
    result
  end
  ◇
```

Macro defined by [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#).
Macro referenced in [1a](#).

Function to apply a transformation to each value of a hash.

```

< definitions 4a > ≡
class Hash
  def apply!(&blk)
    case blk.arity
    when 1
      each_pair{|k,v| self[k]=yield(v)}
    when 2
      each_pair{|k,v| self[k]=yield(k,v)}
    else
      raise ArgumentError.new(' Invalid_arity_of_block ')
    end
  end
  self
end
def apply(&blk)
  h = {}
  case blk.arity
  when 1
    each_pair{|k,v| h[k]=yield(v)}
  when 2
    each_pair{|k,v| h[k]=yield(k,v)}
  else
    raise ArgumentError.new(' Invalid_arity_of_block ')
  end
  h
end
end
◇

```

Macro defined by [2d](#), [4a](#), [6c](#).

Macro referenced in [1a](#).

4.1 Tests

```

< Tests 4b > ≡
def test_accumulate
  assert_equal(66, [11,22,33].accumulate{|x,y| x+y })
  assert_equal(66, [11,22,33].accumulate(0){ |x,y| x+y })
  assert_equal(nil, [].accumulate{|x,y| x+y })
  assert_equal(0, [].accumulate(0){ |x,y| x+y })

  assert_equal(7986, [11,22,33].accumulate{|x,y| x*y })
  assert_equal(7986, [11,22,33].accumulate(1){ |x,y| x*y })
  assert_equal(nil, [].accumulate{|x,y| x*y })
  assert_equal(1, [].accumulate(1){ |x,y| x*y })
end
◇

```

Macro defined by [2c](#), [4bc](#), [5a](#), [6b](#).

Macro referenced in [1d](#).

```

< Tests 4c > ≡
def test_multi_map
  assert_equal ([22,44,66], multi_map([11,22,33]){ |a| 2*a })
  assert_equal ([55,77,99], multi_map ([11,22,33],[44,55,66]) { |a,b| a+b })
  assert_equal ([132,165,198], multi_map ([11,22,33],[44,55,66],[77,88,99]) { |x,y,z| x+y+z })
  assert_equal ([132,165,198], multi_map ([11,22,33],[44,55,66],[77,88,99]) { |x,y| x+y })
  assert_equal([-22,-11,0], multi_map ([11,22,33],[44,55,66],[77,88,99]) { |x,y,z| x+y-z })
end
◇

```

Macro defined by [2c](#), [4bc](#), [5a](#), [6b](#).

Macro referenced in [1d](#).

```

< Tests 5a > ≡
  def test_apply
    assert_equal({'aa'=>22, 'bb'=>44}, {'aa'=>11, 'bb'=>22}.apply{|v| 2*v})
    assert_equal({'aa'=>'aa-11', 'bb'=>'bb-22'}, {'aa'=>11, 'bb'=>22}.apply{|k,v| "#{k}-#{v}"})
    assert_equal({'aa'=>22, 'bb'=>44}, ((h={'aa'=>11, 'bb'=>22});(h.apply!{|v| 2*v});h))
    assert_equal({'aa'=>'aa-11', 'bb'=>'bb-22'}, (h={'aa'=>11, 'bb'=>22};h.apply!{|k,v| "#{k}-#{v}"
};h))
    assert_raise(ArgumentError){h.apply{|x,y,z|}}
    assert_raise(ArgumentError){h.apply!{|x,y,z|}}
    assert_raise(ArgumentError){h.apply{}}
    assert_raise(ArgumentError){h.apply!{}}
  end
  ◇

```

Macro defined by [2c](#), [4bc](#), [5a](#), [6b](#).

Macro referenced in [1d](#).

5 State-Equivalent Classes

This mix-in module by Robert Klemme makes a class's equality-behaviour be based on object state (instance variables).

```

< Goi classes 5b > ≡
  module StateEquivalent
    def ==(obj); test_equal(obj); end
    def eql?(obj); test_equal(obj); end
    def ===(obj); test_equal(obj); end
    def hash
      h = 0
      self.instance_variables.each do |var|
        v = self.instance_eval var
        h ^= v.hash unless v.nil?
      end
      h
    end

    private
    def test_equal(obj)
      return false unless self.class == obj.class
      (self.instance_variables + obj.instance_variables).uniq.each do |var|
        v1 = self.instance_eval var
        v2 = obj.instance_eval var
        return false unless v1 == v2
      end
      true
    end
  end
  ◇

```

Macro referenced in [1a](#).

5.1 Tests

```

< Tests definitions 6a > ≡
  class SEclass
    include StateEquivalent
    def initialize (a,b)
      @a = a
      @b = b
    end
  end
  ◇

```

Macro defined by [2b](#), [6a](#).

Macro referenced in [1d](#).

```

< Tests 6b > ≡
  def test_StateEquivalent
    x = SEclass.new(11,22)
    y = SEclass.new(11,22)
    z = SEclass.new(11,23)
    xx = x
    assert_equal(true,x==xx)
    assert_equal(true,x==y)
    assert_equal(false,x==z)
    assert_equal(x.hash,xx.hash)
    assert_equal(x.hash,y.hash)
    assert_equal(false,x.hash==z.hash)
  end
  ◇

```

Macro defined by [2c](#), [4bc](#), [5a](#), [6b](#).

Macro referenced in [1d](#).

6 Substitution blocks with match data

I don't like having to access globals for the match information in substitution methods, so I'm adding this method:

```

< definitions 6c > ≡
  class String
    # substitution methods accepting a block and passing MatchData
    def gsub_md(pattern)
      gsub(pattern){|str| yield($~)}
    end
    def gsub_md!(pattern)
      gsub!(pattern){|str| yield($~)}
    end
  end
  module Enumerable
    def grep_md(pattern)
      grep(pattern){|str| yield($~)}
    end
  end
  ◇

```

Macro defined by [2d](#), [4a](#), [6c](#).

Macro referenced in [1a](#).

7 Unindent and trim blank lines

This pair of functions are useful when defining a piece of text inside Ruby code with here-documents or other delimitation that includes line breaks. That kind of text is most clearly written with an indentation, and possibly heading or trailing blank lines, which is not desirable when the text is used (e.g. to be print out, or inserted in some file).

La primera elimina el mismo número de espacios en todas las líneas: los que preceden a la primera línea con menor indentación.

```

<Goi functions 7a> ≡
  def un_indent_txt(str)
    s = str.dup
    s.gsub!(/\t/, ' ' * 8)
    mx = s.scan(/^ */).flatten.map{|ss| ss.size}.compact.min
    if mx && mx > 0 then
      re = Regexp.new(' ^_{1, ' + mx.to_s + ' }')
      s = s.gsub(re, "")
    end
    return s
  end
  ◇

```

Macro defined by [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#).
Macro referenced in [1a](#).

Esta función elimina líneas vacías al principio y al final del texto.

```

<Goi functions 7b> ≡
  def trim_lines_txt(str)
    p0 = 0
    pn = str.size
    while (p0 < pn) and /\n\r/.match(str[p0,1])
      p0 += 1
    end
    while (pn > p0) and /\n\r/.match(str[pn-1,1])
      pn -= 1
    end
    str = str[p0 ... pn]
    str
  end
  ◇

```

Macro defined by [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#).
Macro referenced in [1a](#).

```

<Goi functions 7c> ≡
  def flatten_txt(str) # flatten, flush left
    s = "\n" + str
    s = s.gsub(/\n\r +/, "\n")
    s = s[1... s.length]
    return s
  end
  ◇

```

Macro defined by [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#).
Macro referenced in [1a](#).

We'll add another function to eliminate all preceding spaces in every line.

8 Some file utilities

8.1 Relative pathnames

This could be implemented with `pathname` from the standard library:

```

require 'pathname'
Pathname.new(path).relative_path_from(Pathname.new(base)).to_s

```

But this version has more flexibility with lettercase and directory separators (which are normalized by `expand_path`).

```

<Goi functions 7d> ≡

```

```

def relative_path(path, base=nil)
  casefold = < determine if filenames are case insensitive 8a >
  base = opt(base, '.' )
  base = File.expand_path(base)
  base.downcase! if casefold
  base += File::SEPARATOR unless base[-1,1]==File::SEPARATOR
  xpath = File.expand_path(path)
  if xpath.size >= base.size
    pre = xpath[0...base.size]
    pre.downcase! if casefold
    if pre == base
      path = xpath[base.size..-1] # skip separator
      # path = '.' if path=="
    end
  end
end
path
end

```

Macro defined by [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#).
 Macro referenced in [1a](#).

As the Ruby libraries offer no information on whether the running platform system is case-sensitive, we must find out other way. We could use `sys\uname` to detect the running platform, buy we'll use a much more simplistic approach for the time being: if the DOS/Windows path separator is accepted, we assume the system is DOS/Windows and file names are case insensitive; otherwise names are case sensitive.

```

< determine if filenames are case insensitive 8a > ≡
  File::ALT_SEPARATOR=='\\' ◇

```

Macro referenced in [7d](#).

8.2 Copying directories

Copy directories recursively; empty directories are not copied.

```

< Goi functions 8b > ≡
  < copyFiles function 9a >
  ◇

```

Macro defined by [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#).
 Macro referenced in [1a](#).

I've found a problem with `String#sub`: the sequences of two backslashes in the second argument are converted to a single backslash when replaced. To avoid it we always convert path separators to Unix syntax.

```

< copyFiles function 9a > ≡
  def copy_files(from_dir, to_dir, hidden=false, &blk)
    from_dir = File.expand_path(from_dir).gsub(File::ALT_SEPARATOR,File::SEPARATOR) if File::
    ALT_SEPARATOR
    to_dir = File.expand_path(to_dir).gsub(File::ALT_SEPARATOR,File::SEPARATOR) if File::ALT_SEPARATOR
    files = Dir.glob(File.join(from_dir,'**/*'),hidden ? File::FNM_DOTMATCH : 0)
    Dir[from_dir+"/**/*"].each do | from_filename |
      next if hidden && ['..','.'].include?(from_filename)
      is_win_hidden = GOI_WIN32_FILE && File.hidden?(from_filename)
      is_hidden = is_win_hidden || File.basename(from_filename)[0]==?.
      next if !hidden && is_hidden
      to_filename = from_filename.sub(from_dir+"/", to_dir+"/");
      if block_given? # !blk.nil?
        if blk.arity==2
          to_filename = yield(from_filename, to_filename)
        else
          to_filename = yield(to_filename)
        end
      end
      next if to_filename.nil?
      if File.directory?(from_filename)
        FileUtils.mkdir_p to_filename
      else
        new_dir = File.dirname(to_filename);
        FileUtils.mkdir_p new_dir
        if !File.exists?(to_filename) or !FileUtils.cmp(from_filename, to_filename)
          if is_win_hidden
            if File.exists?(to_filename)
              # overwritten hidden file causes error
              File.open(to_filename){|f| f.hidden=false}
            end
          end
          FileUtils.cp from_filename, to_filename, :preserve=>true
          if is_win_hidden
            File.open(to_filename){|f| f.hidden=true}
          end
        end
      end
    end
  end
end
◇

```

Macro referenced in [8b](#), [12](#).

Inicialmente copiábamos manualmente la fecha y hora en lugar de utilizar la opción `preserve`.

```

< copy file timestamp 9b > ≡
  tm = File.mtime(from_filename)
  File.utime tm, tm, to_filename
◇

```

Macro never referenced.

```

< Required Modules 9c > ≡
  require 'fileutils'
◇

```

Macro defined by [1c](#), [9c](#), [10b](#).

Macro referenced in [1a](#).

Para manejar el atributo `hidden` en windows necesitamos un módulo adicional:

```

< Require win32-file 10a > ≡
  begin
    require 'win32/file'
    GOI_WIN32_FILE = true
    < win32-file patches 10c, ... >
  rescue LoadError
    GOI_WIN32_FILE = false
  end
  ◇

```

Macro referenced in [10b](#), [12](#).

```

< Required Modules 10b > ≡
  < Require win32-file 10a >
  ◇

```

Macro defined by [1c](#), [9c](#), [10b](#).

Macro referenced in [1a](#).

The File.dirname in win32-file (at least versions 0.5.2 and 0.5.3) returns "." for any relative path; fix this:

```

< win32-file patches 10c > ≡

  # redefine File .dirname
  def File .dirname(file)
    fpath = false
    file = file .dup

    if file .include?(' / ')
      file .tr !(' / ', "\ ")
      fpath = true
    end

    if PathIsRoot(file)
      file .tr !("\ ", ' / ') if fpath
      return file
    end

    PathRemoveFileSpec(file)
    file = file .split (0.chr) .first || ""
    PathRemoveBackslash(file)

    file .tr !("\ ", ' / ') if fpath
    file = "." if file .empty?
    file
  end
  ◇

```

Macro defined by [10cd](#).

Macro referenced in [10a](#).

Also, win32-file-stat seems to have a problem: lstat raises ArgumentError and SystemCallError is expected...

```

< win32-file patches 10d > ≡

  class FileUtils :: Entry_
    def lstat !
      lstat ()
      rescue SystemCallError, ArgumentError
        nil
      end
    end
  end
  ◇

```

Macro defined by [10cd](#).

Macro referenced in [10a](#).

8.3 UNIX-like tools

These functions need to detect DOS/Windows for special handling of executable files extensions and locations. Since I don't want to make this dependent on `sys/uname`, which would be a good way of detecting the OS, I'll use a simplistic approach:

```
⟨ running on DOS-Windows 11a ⟩ ≡
  File :: ALT_SEPARATOR == '\\ ' ◊
```

Macro referenced in [11b](#).

```
⟨ Goi functions 11b ⟩ ≡
  def which_file(prg, path=ENV['PATH'])
    win = ⟨ running on DOS-Windows 11a ⟩

    path = path.split(File::PATH_SEPARATOR)
    path.unshift '.' if win

    if win
      exts = ENV['PATHEXT']
      exts ||= ".COM; .EXE; .BAT"
      exts = exts.downcase
      exts = exts.split(';')
      if exts.include? File.extname(prg).downcase
        exts = ['']
      end
    else
      exts = ['']
    end

    found = nil
    path.each do |dir|
      exts.each do |ext|
        fn = File.expand_path(File.join(dir, prg + ext))
        if File.exists?(fn) && File.executable?(fn) && !File.directory?(fn)
          found = fn
          break
        end
      end
    end
    found
  end
  ◊
```

Macro defined by [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#).

Macro referenced in [1a](#).

9 installer

```
"install.rb" 12 ≡
  <License 1b>
  require 'rubygems'
  require 'rbconfig'
  require 'fileutils'
  <Require win32-file 10a>

  include Config

  sitelibdir = CONFIG["sitelibdir"]

  from_dir = File.join('goi')
  to_dir = File.join(sitelibdir, 'goi')

  <copyFiles function 9a>

  begin
    copy_files from_dir, to_dir
  rescue
    puts "Error installing goitools: _[" + $! + "]"
    exit
  end

  puts "goitools_has_been_successfully_installed"
  ◇
```

10 Índices

10.1 Archivos

"goi/tools.rb" Defined by [1a](#).
 "install.rb" Defined by [12](#).
 "tools_test.rb" Defined by [1d](#).

10.2 Fragmentos

<Goi classes [5b](#)> Referenced in [1a](#).
 <Goi functions [2a](#), [3](#), [7abcd](#), [8b](#), [11b](#)> Referenced in [1a](#).
 <License [1b](#)> Referenced in [1ad](#), [12](#).
 <Require win32-file [10a](#)> Referenced in [10b](#), [12](#).
 <Required Modules [1c](#), [9c](#), [10b](#)> Referenced in [1a](#).
 <Tests definitions [2b](#), [6a](#)> Referenced in [1d](#).
 <Tests [2c](#), [4bc](#), [5a](#), [6b](#)> Referenced in [1d](#).
 <classes ?> Referenced in [1a](#).
 <copy file timestamp [9b](#)> Not referenced.
 <copyFiles function [9a](#)> Referenced in [8b](#), [12](#).
 <definitions [2d](#), [4a](#), [6c](#)> Referenced in [1a](#).
 <determine if filenames are case insensitive [8a](#)> Referenced in [7d](#).
 <running on DOS-Windows [11a](#)> Referenced in [11b](#).
 <win32-file patches [10cd](#)> Referenced in [10a](#).

10.3 Identificadores