

1 Configuration Level 0

This Ruby module provides a platform-independent way of accessing basic configuration data with the following features:

- Configuration data is kept in text form (YAML format) so it can be easily edited; it can be optionally stored in the Windows Registry.
- Data can be unique per machine or different data can be assigned to different users (not on Windows 9x), also data can be related to the executable program installation.

The intended use of this module is managing the minimal information to access the rest of the configuration. For example configuration could be stored in a database, but the program needs some data to be able to access the database (which database server or files, etc.). So this provides a first level of configuration, and hence the name.

```
"goi/cfg0.rb" 1a ≡
  # Configuration Level 0.
  # Simple application configuration data, user or host specific .
  <License 1b>
  <Required Modules 1c, ... >
  #Cfg0 objects give access to the configuration data.
  <Examples of use 9d, ... >
  module Goi
    class Cfg0
      <definitions 1d, ... >
      private
      <private methods 7a, ... >
    end
  end
  end
  ◇
```

```
<License 1b> ≡
  # Copyright (C) 2004, 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 [1a](#).

2 Needed Modules

2.0.1 uname

We'll use `uname` to be able to identify the system we're running in. This will be needed because different systems use different file systems, and the Registry is available only on Windows systems.

This library is provided as functions within a module, so it must be extended into some object or included in a class; we'll include the definitions in our class.

```
<Required Modules 1c> ≡
  require 'sys/uname'; ◇
```

Macro defined by [1c](#), [2ab](#).

Macro referenced in [1a](#).

```
<definitions 1d> ≡
  include Sys
  ◇
```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).

Macro referenced in [1a](#).

2.0.2 fileutils

We require `fileutils` for some file system operations.

```
< Required Modules 2a > ≡
  require 'fileutils';◇
```

Macro defined by [1c](#), [2ab](#).

Macro referenced in [1a](#).

2.0.3 YAML

We'll use the YAML format to store our configuration data.

```
< Required Modules 2b > ≡
  require 'yaml';◇
```

Macro defined by [1c](#), [2ab](#).

Macro referenced in [1a](#).

3 Initialization

A `Cfg0` object will represent the configuration for an application, that is, a set of values indexed by keys.

The object will be defined by the type of data (unique per user, `:user`, per machine, `:host` or per installation base), an identifier (text string) for the application, an optional group name (to have configurations of related applications together in a sub-directory), and an optional flag to use the Windows Registry.

```
< definitions 2c > ≡
  # Create Configuration Object for an applicatin
  # - mode is either :user, :host or :base
  # - appid is an application identifier (name)
  # - group is an optional group name
  # - use_reg is a flag to use the Windows Registry (if available )
  def initialize (mode, appid, group=nil, use_reg=true)
    < initialization 2d, ... >
  end
  ◇
```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).

Macro referenced in [1a](#).

On creation we first store the defining parameters in instance variables.

```
< initialization 2d > ≡
  @mode = mode # :user :host :base
  @appid = appid
  @group = group
  @use_reg = use_reg && /Windows .. /.match(Unicode.sysname)
  ◇
```

Macro defined by [2d](#), [3a](#), [4d](#).

Macro referenced in [2c](#).

Then we will compute and keep the information needed to store the data. Note that the `:base` mode is somewhat different; it involves the `running` script; other modes may refer to an application different from the current one.

```

< initialization 3a > ≡
  case @mode
  when :user
    if /Windows ../.match(Uname.sysname)
      < compute per-user configuration file location for Windows 3b >
    else
      < compute per-user configuration file location for other systems 4a >
    end
  when :host
    if /Windows ../.match(Uname.sysname)
      < compute per-host configuration file location for Windows 4b >
    else
      < compute per-host configuration file location for other systems 4c >
    end
  when :base
    @file_dir = Cfg0.base_dir
  else
    # mode is a directory name
    @file_dir = mode.to_str
  end
  use_dot = !/Windows ../.match(Uname.sysname) && mode==:user
  if @group
    group_dir = (use_dot ? ' .' : '') + @group
    use_dot = false
    @file_dir = File .join (@file_dir, group_dir)
  end
  @file_dir = @file_dir.gsub(Regexp.new(Regexp.escape(File::ALT_SEPARATOR)), File::SEPARATOR) unless File::
  ALT_SEPARATOR.nil?
  @file_name = appid+ '.cfg'
  @file_name = ' .' +@file_name if use_dot
  ◇

```

Macro defined by [2d](#), [3a](#), [4d](#).

Macro referenced in [2c](#).

We'll do some reasonable effort to use sensible directories under different Windows versions (excluding versions as older as Windows NT 3.x or Windows 3.x). For a more conforming solution, the shell32.dll API should be used.

```

< compute per-user configuration file location for Windows 3b > ≡
  if Uname.sysname=="Windows_NT"
    if Uname.release[0]>=?5
      # Windows 2000, XP, 2003, .... (its reported not writable by File.writable? but it is writable)
      @file_dir = ENV['APPDATA']
    else
      # Windows NT
      @file_dir = File .join (ENV['USERPROFILE'], "Application_Data") # assume no language-dependent, e.g. "
      Datos de Programa"
      # ENV['USERPROFILE'] = ENV['HOMEDRIVE']+ENV['HOMEPATH']
    end
  else
    @file_dir = File .join (File .join (File .join (ENV['WINDIR'], 'Profiles'), ENV['USERNAME']), "Application
    _Data")
  end
  ◇

```

Macro referenced in [3a](#).

Note: it would be interesting to have this or other utility module to determine the home path for non-unix systems and store it in ENV['HOME'] (that way File.expand_path would expand ~ correctly).

Hopefully, all other systems will be Unix-like.

```

< compute per-user configuration file location for other systems 4a > ≡
  @file_dir = ENV['HOME']
  ◇

```

Macro referenced in [3a](#).

For the `:host` mode we need to select a well-defined location in the local filesystem, that is writable.

```

< compute per-host configuration file location for Windows 4b > ≡
  if Uname.sysname=="Windows_NT"
    if Uname.release[0]>=?5
      # Windows 2000, XP, 2003, ...
      @file_dir = File.join(ENV['ALLUSERSPROFILE'],File.basename(ENV['APPDATA']))
      # this is probably not writable for common users, though ...
      # --wait, no, it is reported non-writable but it is!
      if !File.writable?(@file_dir) && !use_reg
        # to do: find a suitable, writable location ...
      end
    end
  else
    # Windows NT
    @file_dir = File.join(ENV["SystemRoot"], 'Profiles/All_Users/Application_Data')
  end
else
  # this is commonly used by many applications, although not the Microsoft way of doing it
  @file_dir = File.join(ENV['windir'],'Application_Data')
end
◇

```

Macro referenced in [3a](#).

Again, we expect non-Windows systems to be Unix-like. Here we need a location which is machine-specific (unshared), writable, separate from the system-distribution files (local), and also related to user applications rather than to system applications. I think we should use `/var/local/etc` to comply with all this, though applications more often use `/usr/local/etc` (which is not FHS-compliant, what about `/usr/local/share/etc`?) Other locations could be `/usr/local/share/etc` or `/etc/goi-cfg0` (but the latter may not be writable).

```

< compute per-host configuration file location for other systems 4c > ≡
  @file_dir = '/var/local/etc'
  ◇

```

Macro referenced in [3a](#).

If use of the Windows Registry is required and available we must also find suitable locations in it. Even when the registry is used, existing text files will have priority (so that there's an easy way to change configuration without editing the Registry). Note that `:base` would be treated like `:host` in the Registry.

```

< initialization 4d > ≡
  if @use_reg
    if ["Windows_NT","Windows_9x","Windows_CE"].include?(Uname.sysname)
      require 'win32/registry'
      @reg_base = @mode==:user ? Win32::Registry::HKEY_CURRENT_USER : Win32::Registry::
HKEY_LOCAL_MACHINE
      @reg_node= "Software"
      @reg_node += "\\ "+group if group
      @reg_node += "\\ "+appid
    end
  end
  ◇

```

Macro defined by [2d](#), [3a](#), [4d](#).

Macro referenced in [2c](#).

4 Interface

First we'll provide accessors for the defining attributes of the configuration.

```

< definitions 5a > ≡
  attr_reader :mode, :appid, :group
  # is the Windows registry used instead of configuration files ?
  def registry?; @use_reg; end
  ◇

```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
 Macro referenced in [1a](#).

The interface provides a way to store and retrieve values indexed by key values. If the Registry is to be used, it will have a lower priority than a file if it exists. We'll delegate the actual read and write operations to private methods.

```

< definitions 5b > ≡
  #Read configuration value giving its key
  def read(key)
    value = file_read(key)
    value = reg_read(key) if @use_reg && !value
    value
  end
  ◇

```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
 Macro referenced in [1a](#).

```

< definitions 5c > ≡
  #Write configuration value giving its key
  def write(key, value)
    if @use_reg
      if File.exist?(File.join(@file_dir, @file_name))
        file_write(key,value)
      else
        reg_write(key,value)
      end
    else
      file_write(key,value)
    end
  end
  self
end
  ◇

```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
 Macro referenced in [1a](#).

We'll provide also an alternate array-like (indexed) interface.

```

< definitions 5d > ≡
  #Read configuration value giving its key
  def [](key); read(key); end
  #Write configuration value giving its key
  def []=(key,value); write(key,value); end
  ◇

```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
 Macro referenced in [1a](#).

We'll add access to the directory that would be used for the configuration files (even if the registry is being used).

```

< definitions 5e > ≡
  def dir
    @file_dir
  end
  ◇

```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
 Macro referenced in [1a](#).

4.1 Running Script

We'll also give access with this class to the directory where the script resides and to the full name of the running script. The base directory is a good place to find files that accompany the application.

Note that `Dir.pwd` may not be the directory where the script is installed (this depends on how the script is executed), but in that case `$0` will contain either an absolute full path or a path relative to it, so this is not problem.

No! this may fail to be so if a Windows' shortcut is used to launch the program.

```
<definitions 6a> ≡
  # full name of the executing script
  def Cfg0.exe_file
    File.expand_path($0,Dir.pwd)
  end
  ◇
```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
Macro referenced in [1a](#).

```
<definitions 6b> ≡
  # directory from where the script is installed
  def Cfg0.base_dir
    File.dirname(Cfg0.exe_file)
  end
  ◇
```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
Macro referenced in [1a](#).

Although currently a `Cfg0` does not keep any resources open, we will provide the typical Ruby `open` command to have access to a `Cfg` from a block of code.

```
<definitions 6c> ≡
  # create a configuration and access it from a block
  def Cfg0.open(mode, appid, group=nil, use_reg=true)
    cfg = Cfg0.new mode, appid, group, use_reg
    yield cfg
  end
  ◇
```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
Macro referenced in [1a](#).

4.2 Versions

We'll add a method to support application versions. Application version should be appended as a prefix to the application id, like in `application-1.0.0`, if different versions of the same application can be installed in the same system.

We'll provide a `versions` method intended to be used with a `:base` configuration, to find all versions available of a given application. For generality, a pattern will be passed for the application identifier, so that all matching applications are returned.

```
<definitions 6d> ≡
  # create all configurations matching an application -id pattern (versions)
  def Cfg0.open_all(appid, group=nil, use_reg=true)
    cfg = Cfg0.new :host, appid, group, use_reg
    cfg.versions.each do |ver|
      ver_cfg = Cfg0.new(:host, ver, group, use_reg)
      yield ver_cfg
    end
  end
  ◇
```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
Macro referenced in [1a](#).

5 Implementation

We will now define the methods to do the actual reading and writing.

5.1 Configuration files

```

< private methods 7a > ≡
  def file_read(key)
    fn = File.join(@file_dir, @file_name)
    file = {}
    # file = YAML::load(File.open(fn))
    File.open(fn){|f| file = YAML::load(f) if File.exists?(fn)
    file[key]
  end
  ◇

```

Macro defined by [7abc, 8a](#).

Macro referenced in [1a](#).

```

< private methods 7b > ≡
  def file_write(key, value)
    FileUtils.mkdir_p @file_dir if !File.exist?(@file_dir)
    fn = File.join(@file_dir, @file_name)
    file = {}
    # file = YAML::load(File.open(fn))
    File.open(fn){|f| file = YAML::load(f) if File.exists?(fn)
    file[key] = value
    File.open(fn,"w"){|f| f << file.to_yaml}
  end
  ◇

```

Macro defined by [7abc, 8a](#).

Macro referenced in [1a](#).

5.2 The Windows Registry

```

< private methods 7c > ≡
  def reg_read(key)
    begin
      @reg_base.open(@reg_node) do |reg|
        key_value = reg.read(key)[1]
        if key_value[0...3]== '---'
          value = YAML::load(key_value)
        else
          # read non YAML string to share values with other apps ...
          value = key_value
        end
      end
    rescue
      nil
    end
  end
  ◇

```

Macro defined by [7abc, 8a](#).

Macro referenced in [1a](#).

```

<private methods 8a> ≡
  def reg_write(key,value)
    # don't use yaml for string, for compatibility with other apps ...
    value = value.respond_to?(:to_str) ? value.to_str : value.to_yaml
    @reg_base.create(@reg_node) do |reg|
      reg[key] = value
    end
  end
end
◇

```

Macro defined by [7abc](#), [8a](#).
 Macro referenced in [1a](#).

5.3 Handling Versions

This method obtains an application available versions, i.e., the application identifiers for those applications installed in the system (i.e. that have stored some per-host configuration data) that match a given pattern. By default the pattern uses the syntax of `glob`, but any regular expression can be used as well.

```

<definitions 8b> ≡
  # Obtains all versions of an applications, i.e.
  # the application identifiers of installed applications (with some :host data)
  # that match the given pattern. The pattern uses the special characters of glob.
  def versions(glob_patt=true)
    vers = []
    # @appid is treated as a pattern
    patt = @appid
    patt = Regexp.new(@appid) if !glob_patt
    if @use_reg
      # remove appid (last component) from @reg_node
      base_key = @reg_node.split('\\')[0...-1].join('\\')
      if glob_patt
        <convert glob pattern to regular expression 9c>
      end
      <collect versions from the registry 8c>
    else
      base_dir = @file_dir
      if glob_patt
        <collect versions from files with glob syntax 9a>
      else
        <collect versions from files by reg.exp. 9b>
      end
    end
    vers
  end
end
◇

```

Macro defined by [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#).
 Macro referenced in [1a](#).

Here we iterate over the applications of the group in the registry, and keep matching identifiers. We use a regular expression for the pattern.

```

<collect versions from the registry 8c> ≡
  @reg_base.open(base_key) do |reg|
    reg.each_key do |key, wtime|
      if patt.match(key)
        vers << key
      end
    end
  end
end
◇

```

Macro referenced in [8b](#).

```

< collect versions from files with glob syntax 9a > ≡
  Dir.glob(File.join(@file_dir,@file_name)) do |file|
    if !File.directory?(file)
      vers << File.basename(file,'.cfg')
    end
  end
end
◇

```

Macro referenced in [8b](#).

```

< collect versions from files by reg.exp. 9b > ≡
  Dir.foreach(base_dir) do |file|
    if patt.match(file)
      if !File.directory?(File.join(base_dir, file))
        vers << File.basename(file,'.cfg')
      end
    end
  end
end
◇

```

Macro referenced in [8b](#).

Conversion of glob pattern to regular expression. Currently doesn't support **

```

< convert glob pattern to regular expression 9c > ≡
  patt = Regexp.escape(patt)
  patt.gsub!(/\\*\/,'.*')
  patt.gsub!(/\\?\/,'.')
  patt.gsub!(/\\\[^\]\\]*\\\/){|alts| ' ('+alts[2...-2].split(' ').join(' ') | (')+' )' }
  patt.gsub!(/\\\[^\]\\]*\\\/){|alts| '['+alts[2...-2]+' ]' }
  patt = Regexp.new patt
◇

```

Macro referenced in [8b](#).

6 Example

```

< Examples of use 9d > ≡
#
#To handle user-specific configuration for an application named 'an-app',
#we would create an object:
# require 'goi/cfg0'
# cfg = Cfg0.new(:user, 'an-app')
#And we can then store configuration variables like this
# cfg['un valor']='/ directorio /base'
# cfg['otro valor']=3433.2
#And retrieve them: (nil will be returned for non-existing values)
# dir_base = cfg['un valor']
# x = cfg['otro valor']
#To create a non-user-specific configuration we would do:
# cfg = Cfg0.new(:host, 'an-app')
#If we want to keep several configuration files together in a
#directory (e.g. for related applications) we can use a group name:
# cfg = Cfg0.new(:host, 'an-app', 'a-group')
#In Windows, the Registry will be used by default instead of configuration
#files, but this can be disabled like this:
# cfg = Cfg0.new(:host, 'an-app', 'a-group', false)
#Note: even if the Registry is used, we can override it with a
#manually created file.
#◇

```

Macro defined by [9de](#).

Macro referenced in [1a](#).

```

< Examples of use 9e > ≡
#
#Cfg0 can be used to query applications other than the one it is called from,
#but not for :base data, which always refers to the running application .
#For example, we could do this to call another application if it is installed :
# Cfg0.open(:host, 'other-app', 'other-group') do |cfg|
#   if cfg['Version']
#       # the other application is installed
#       if cfg['Version']>=needed_version
#           system cfg['Command']
#       end
#   end
# end
#For this to work an application should contain code like this :
# Cfg0.open(:host, 'this-app', 'group') do |cfg|
#   if cfg['Version']<VERSION
#       cfg['Version'] = VERSION
#       cfg['BaseDir'] = Cfg0.base_dir
#       cfg['Command'] = Cfg0.exe_file+' '+arguments
#   end
# end
#But that won't work if the application is converted to a native executable
#with rubyscript2exe; in that case the installation process should set the
#correct BaseDir and Command.
#If we want to use the latest version of another applicaton we could do this :
# vers = {}
# Cfg0.open_all('other-app-*','other-group'){ |cfg| vers[cfg['Version']] = cfg }
# ver, cfg = vers.max
# if ver>=needed_version
#   system cfg['Command']
# end
#Or we could have done this :
# other_app_ver = Cfg0.open(:host, 'other-app-*','other-group'){ |cfg| cfg.versions }.max
# Cfg0.open(:host, other_app_ver, 'other-group') ...
#◇

```

Macro defined by [9de](#).

Macro referenced in [1a](#).

7 Índices

7.1 Archivos

"goi/cfg0.rb" Defined by [1a](#).

7.2 Fragmentos

< Examples of use [9de](#) > Referenced in [1a](#).

< License [1b](#) > Referenced in [1a](#).

< Required Modules [1c](#), [2ab](#) > Referenced in [1a](#).

< collect versions from files by reg.exp. [9b](#) > Referenced in [8b](#).

< collect versions from files with glob syntax [9a](#) > Referenced in [8b](#).

< collect versions from the registry [8c](#) > Referenced in [8b](#).

< compute per-host configuration file location for Windows [4b](#) > Referenced in [3a](#).

< compute per-host configuration file location for other systems [4c](#) > Referenced in [3a](#).

< compute per-user configuration file location for Windows [3b](#) > Referenced in [3a](#).

< compute per-user configuration file location for other systems [4a](#) > Referenced in [3a](#).

< convert glob pattern to regular expression [9c](#) > Referenced in [8b](#).

< definitions [1d](#), [2c](#), [5abcde](#), [6abcd](#), [8b](#) > Referenced in [1a](#).

< initialization [2d](#), [3a](#), [4d](#) > Referenced in [2c](#).

< private methods [7abc](#), [8a](#) > Referenced in [1a](#).

7.3 Identificadores