

## 1 Applicaton Logging

This module provides a mechanism to log application events and errors in the debugging and production stages. Application messages are saved in a temporary file that is made persistent in case of error. In production code, the kept log files are packed in a .zip file so they can easily transmitted to the developers.

```
"goi/applog.rb" 1a ≡
  # Application error logging.
  <License 1b>
  <Required Modules 1c>
  #AppLog objects provide logging
  <Examples of use 9b>
  module Goi
    class AppLog
      <definitions 1d,...>
      private
      <private methods 3a,...>
    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](#), [9a](#).

## 2 Needed Modules

```
<Required Modules 1c> ≡
  require 'goi/cfg0'
  require 'goi/tools'
  require 'goi/zipx'
  require 'tmpdir'
  require 'pathname'
  require 'zip/zipfilesystem'
  ◇
```

Macro referenced in [1a](#).

## 3 Initialization

```
<definitions 1d> ≡
  # Create an AppLog object; it will contain also Cfg0 objects.
  # -id is the application identifier (name)
  # -gr is an optional group name
  # -reg is a flag to use the Windows Registry (if available)
  def initialize (id, gr=nil,reg=true)
    <initialization 2b>
  end
  end
  ◇
```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
Macro referenced in [1a](#).

```

< definitions 2a > ≡
  attr_reader :appid, :group, :host_cfg, :user_cfg, :base_cfg
  ◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
Macro referenced in [1a](#).

```

< initialization 2b > ≡
  @appid = id
  @group = gr
  @host_cfg = Cfg0.new(:host,id,gr,reg)
  @user_cfg = Cfg0.new(:user,id,gr,reg)
  @base_cfg = Cfg0.new(:base,id,gr,reg)

  @tmp_dir = Dir.tmpdir
  @host_dir = @host_cfg.dir
  @user_dir = @user_cfg.dir
  @base_dir = Cfg0.base_dir

  @msgout = nil
  @prgout = nil

  @is_open = false
  @is_persistent = false

  today = Time.now.strftime('%Y%m%d')
  @log_fn = uniq_fn(@tmp_dir, "#{id}_#{today}", ".log")

  < determine a writable directory suitable for the zip file 2c >
  @zip_fn = File.join(wr_dir, "ERRORES.zip") # @host_dir ?

  @keep_open = true
  @log_file = nil
  ◇

```

Macro referenced in [1d](#).

When an error occurs the log file will be packed into a zip file that the user can send to the developers. This file should be in an easy place to be found, such as the same location of the executed script, but it needs to be writable. Alo, we'll take care not to use the program location when it is a temporary location created by `rubyscript2exe`.

Note that the Windows host and user directories are reported non-writable so they are not used. They're probably writable, but they're not good candidates anyway, because they're behind hidden directories.

```

< determine a writable directory suitable for the zip file 2c > ≡
  wr_dir = @base_dir
  if !File.writable?(wr_dir) || is_exe_tmp(wr_dir)
  # wr_dir = Dir.pwd
  # if !File.writable?(wr_dir) || is_exe_tmp(wr_dir)
  wr_dir = @host_dir
  if !File.writable?(wr_dir) || is_exe_tmp(wr_dir)
  wr_dir = @user_dir
  if !File.writable?(wr_dir) || is_exe_tmp(wr_dir)
  wr_dir = @tmp_dir
  end
  end
  # end
  end
  ◇

```

Macro referenced in [2b](#).

This method tries to detect a temporary location as used by `rubyscript2exe`. This may fail for future versions on that program.

```

<private methods 3a> ≡
  #This tries to detect a rubyscript2exe temporary directory
  def is_exe_tmp(fn)
    is = false
    if Pathname.new(fn).relative_path_from(Pathname.new(@tmp_dir)).to_s[0,2]!='..'
      b = File.basename(fn)
      b = File.basename(File.dirname(fn)) if b=='app'
      if b.match(/eee\.\d+/)
        is = true
      end
    end
  end
  is
end
◇

```

Macro defined by [3ab](#), [5b](#).

Macro referenced in [1a](#).

This will compute nonexisting file names.

```

<private methods 3b> ≡
  def uniq_fn(dir, prefix, ext)
    fn = File.join(dir, prefix)
    suffix = ''
    if File.exists?(fn+ext)
      suffix = '_001'
      suffix = suffix.succ while File.exists?(fn+suffix+ext)
    end
    fn += suffix + ext
  end
  fn
end
◇

```

Macro defined by [3ab](#), [5b](#).

Macro referenced in [1a](#).

## 4 Methods

```

<definitions 3c> ≡
  #Open the log file
  def open
    close if @is_open
    if @keep_open
      @log_file = File.new(@log_fn, "a")
    end
    @is_open = true

    msg = %Q{< Application Information 4b >}

    log(msg)

  end
  ◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).

Macro referenced in [1a](#).

```

< definitions 4a > ≡
  #Close the log file
  def close
    if @is_open
      if @keep_open
        @log_file.close
      end
      if !@is_persistent
        if File.exist?(@log_fn)
          File.delete(@log_fn)
        end
      end
      @is_open = false
    end
  end
  ◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
 Macro referenced in [1a](#).

```

< Application Information 4b > ≡
  Inicio de #{group ? group+'/' +appid : appid}
  TmpDir: #{@tmp_dir}
  UserDir: #{@user_dir}
  HostDir: #{@host_dir}
  BaseDir: #{@base_dir}
  System: #{Sys::Uname.sysname}
  Version: #{Sys::Uname.release}
  Build: #{Sys::Uname.version}
  Machine: #{Sys::Uname.machine}
  Node: #{Sys::Uname.nodename}
  Ruby Version: #{RUBY_VERSION}
  Ruby Platform: #{RUBY_PLATFORM}
  Program Name: #{ $0 }
  Directory: #{Dir.pwd}
  ◇

```

Macro referenced in [3c](#).

```

< definitions 4c > ≡
  #Make the log file persistent (do not delete it at closing)
  def persist
    @is_persistent = true
  end
  ◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
 Macro referenced in [1a](#).

This is the method to record information in the log file.

```

<definitions 5a> ≡
  #Send information to the log file : either an exception object or a text .
  def log(txt, timestamp=true, show=false)
    if @is_open
      txt = errorMsg(txt)
      showProgressMsg(txt) if show
    if @keep_open
      do_log(@log_file, txt, timestamp)
    else
      File.open(@log_fn, "a") do |file|
        do_log(file, txt, timestamp)
      end
    end
  end
end
end
◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
 Macro referenced in [1a](#).

```

<private methods 5b> ≡
  def do_log(file, txt, timestamp=true)
    if timestamp
      file.puts Time.now.strftime('%Y-%m-%d_%H:%M:%S_-----\n')
    end
    file.puts txt
    file.puts "-----\n"
    file.flush
  end
◇

```

Macro defined by [3ab](#), [5b](#).  
 Macro referenced in [1a](#).

Sometimes we may want to put a message in the log file and also display it to the user to show progress...

```

<definitions 5c> ≡
  #Send information to the log file : either an exception object or a text .
  def log_show(txt, timestamp=true)
    log txt, timestamp, true
  end
◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
 Macro referenced in [1a](#).

Each information to be written to the log is processed by this method: it accepts either exception objects or text. For text, blank lines at the start or end are removed, and the text is un-indented.

```

<definitions 5d> ≡
  #Prepare a message to be logged or shown.
  def errorMsg(txt, timestamp=true)
    if txt.kind_of?(Exception) # Exception or StandardError ?
      txt = "Error: _#{txt.message}\n[error_class: _#{txt.class}]\n" + txt.backtrace.join("\n")
    end
    txt = Goi::trim_lines_txt(Goi::un_indent_txt(txt))
    txt
  end
◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
 Macro referenced in [1a](#).

This methods displays a message to the user, rather than logging it.



```

< definitions 7a > ≡
  #Email account to send error reports to.
  def email
    em = base_cfg['ErrEmail']
    em = host_cfg['ErrEmail'] unless em
    em = 'informativa@einsl.com' unless em
    em
  end
  ◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
 Macro referenced in [1a](#).

These methods return the name of the log file and of the zip file where log files may be packed.

```

< definitions 7b > ≡
  #log file name
  def filename
    @log_fn
  end
  #zip file name (where log files are packed)
  def zipname
    @zip_fn
  end
  ◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
 Macro referenced in [1a](#).

This method packs the log file into a zip file (which may contain previous log files of the same name which are preserved).

```

< definitions 7c > ≡
  #pack log file into zip file preserve previous log even if they have the same name
  def zip
    return unless @is_open
    if @keep_open
      @keep_open = false
      @log_file.close
    end
    create = File.exist?(zipname) ? nil : Zip::ZipFile::CREATE
    zpnm = File.basename(filename)
    Zip::ZipFile.open(zipname, create) do |zf|
      unless create
        suffix = nil
        prefix,*ext = zpnm.split('.')
        while zf.file.exists?(zpnm)
          if suffix
            suffix.succ!
          else
            suffix = '_01'
          end
          zpnm = [prefix + suffix].concat(ext).join('.')
        end
        zf.add(zpnm, filename){true}
        #@is_persistent = false
      end
    end
  end
  ◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
 Macro referenced in [1a](#).

## 5 Application execution wrapper

This method should be use to run an application code that uses logging. This will preserve the log file in case of error (and will include information about the error in the log). Also, if debug mode is not selected, the preserved log will be packed into a zip file and instructions will be given to the user to send the zip file to the developers.

```

< definitions 8a > ≡
  #Execute the application code, providing an AppLog object,
  #and handling errors.
  #A block must be passed.
  def AppLog.run(id,options={})
    gr = opt(options[:group],nil)
    reg = opt(options[:registry ],true)
    msgout = opt(options[:msgout],nil)

    log = AppLog.new(id,gr,reg)
    log.setMsgOut(msgout)
    log.open
    yield(log)

    rescue StandardError=>err
      log.log err
      if $DEBUG
        #message to the tester /developer
        log.persist
        log.showMsg(err, "Log_file:_{#{log.filename}")
      else
        #message to the unsuspecting user
        log.zip
        txt = %Q{< Error Message 8b >}
        log.showMsg(txt)
      end

    ensure
      log.close
    end
  end
  ◇

```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).

Macro referenced in [1a](#).

```

< Error Message 8b > ≡
  Ha ocurrido un error.
  La información necesaria para analizar sus causas
  se encuentra en el archivo:
    #{log.zipname}
  Por favor, envíe dicho archivo a #{log.email}.
  Añada cualquier comentario que considere útil
  sobre las circunstancias en las que ha ocurrido el error.
  Gracias.
  ◇

```

Macro referenced in [8a](#).

## 6 Tests

```
"test_applog.rb" 9a ≡
<License 1b>
require 'goi/applog'
include Goi
def test_msg(txt)
  puts "\n"
  puts "======"
  txt.each{|line| puts "=====" +line}
  puts "======"
  puts "\n"
end

def test_applog
  AppLog.run('testal') do |app|
    #app.setMsgOut proc{|txt| test_msg(txt)}
    app.setMsgOut(method(:test_msg))
    app.log "App._Version:#{app.host_cfg['Version']}"
    app.log "App._BaseDir:#{app.host_cfg['BaseDir']}"
    app.log "App._Command:#{app.host_cfg['Command']}"
    # app.log "XXXXXX:#{app.user_cfg['XXXXXX']}"
    x = 3
    app.log "vamos_a_dividir"
    x /=_0
  end
end

test_applog
◇
```

```
<Examples of use 9b> ≡
# require 'goi/applog'
# AppLog.run('test_app') do |app|
#   app.setMsgOut proc{|txt| ...}
#   app.log "App. Version: #{app.host_cfg ['Version']}"
#   ...
#   app.log "..."
#   ...
# end
◇
```

Macro referenced in [1a](#).

## 7 Configuración en cascada

Estos métodos permiten leer valores de configuración “en cascada”, dando prioridad a la configuración particular del usuario en primer lugar, luego a la de la máquina y por último a la asociada al directorio de instalación.

```
<definitions 9c> ≡
#Read configuration value giving its key
def cfg_read(key)
  value = user_cfg[key]
  value = host_cfg[key] unless value
  value = base_cfg[key] unless value
  value
end
◇
```

Macro defined by [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#).  
Macro referenced in [1a](#).

## 8 Índices

### 8.1 Archivos

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

"test\_applog.rb" Defined by [9a](#).

### 8.2 Fragmentos

⟨ Application Information [4b](#) ⟩ Referenced in [3c](#).

⟨ Error Message [8b](#) ⟩ Referenced in [8a](#).

⟨ Examples of use [9b](#) ⟩ Referenced in [1a](#).

⟨ License [1b](#) ⟩ Referenced in [1a](#), [9a](#).

⟨ Required Modules [1c](#) ⟩ Referenced in [1a](#).

⟨ definitions [1d](#), [2a](#), [3c](#), [4ac](#), [5acd](#), [6abcd](#), [7abc](#), [8a](#), [9c](#) ⟩ Referenced in [1a](#).

⟨ determine a writable directory suitable for the zip file [2c](#) ⟩ Referenced in [2b](#).

⟨ initialization [2b](#) ⟩ Referenced in [1d](#).

⟨ private methods [3ab](#), [5b](#) ⟩ Referenced in [1a](#).

### 8.3 Identificadores