RubyScript2Exe - A Ruby Compiler

Table of Contents

RubyScript2Exe

A Ruby Compiler

Tue May 29 20:09:00 UTC 2007

Erik Veenstra <rubyscript2exe@erikveen.dds.nl>

1. Introduction

RubyScript2Exe transforms your Ruby application into a standalone, compressed Windows, Linux or Mac OS X (Darwin) executable. You can look at it as a "compiler". Not in the sense of a source-code-to-byte-code compiler, but as a "collector", for it collects all necessary files to run your application on an other machine: the Ruby application, the Ruby interpreter and the Ruby runtime library (stripped down for your application). Anyway, the result is the same: a standalone executable (`application.exe`). And that's what we want!

Because of the gathering of files from your own Ruby installation, RubyScript2Exe creates an executable for the platform it's being run on. No cross compile.

And when I say Windows, I mean both Windows (RubyInstaller, MinGW and MSWin32) and Cygwin. But the generated `exe` under Cygwin is very, very big, because its `exe`'s are very big (static?) and it includes `cygwin1.dll`, so it can run on machines without Cygwin.

There is one more advantage: Because there might be some incompatibilities between the different Ruby versions, you have to test your application with every single version. Unless you distribute your version of Ruby with your application...

RubyScript2Exe can handle simple scripts, but it can handle complete directories as well. Usually, an application is more than just a program or a script. It consists of libraries, documentation, help files, configuration files, images, licenses, readmes, and so on. You can embed all of them in one single executable.

What's the difference between RubyScript2Exe and AllInOneRuby? Well, RubyScript2Exe includes an application (your script), the Ruby VM and only parts of the ruby_lib tree (it's stripped specifically for your application). AllInOneRuby contains a complete Ruby installation: it includes no application, but it does include the Ruby VM and the complete ruby_lib tree. You can use `allinoneruby.exe` like `ruby.exe` (Windows) and `allinoneruby_*` like `ruby` (Linux, Darwin) that's already installed on your system. In other words: the executable, generated with RubyScript2Exe, is an application; the one generated with AllInOneRuby "is" Ruby.

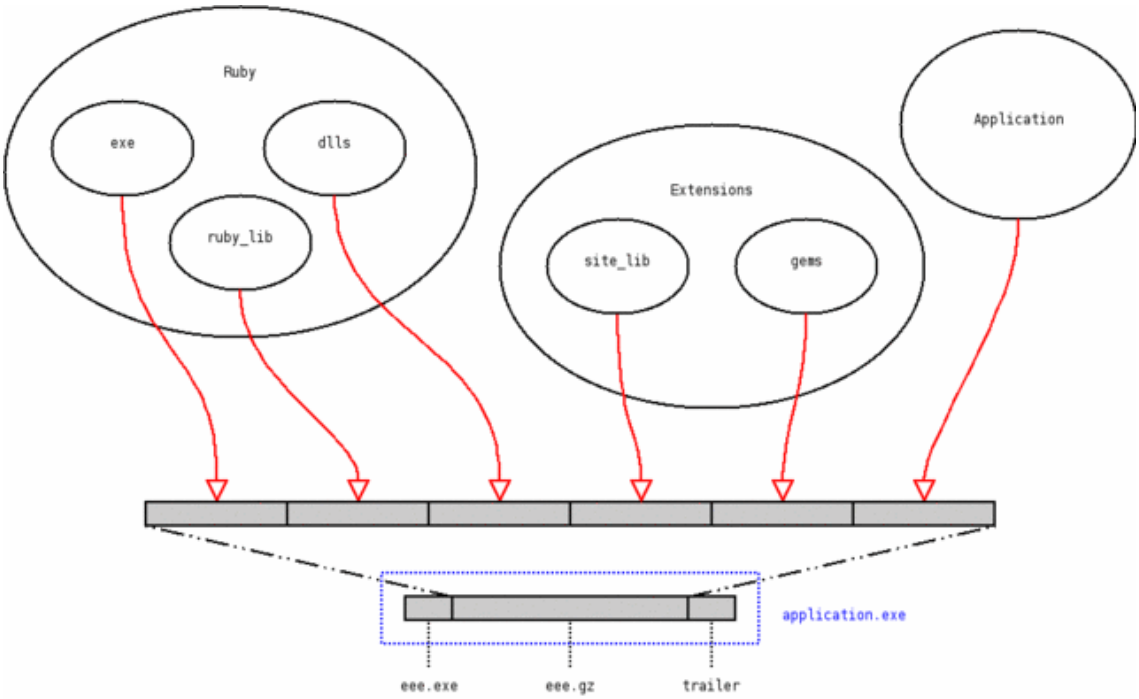
If you like RubyScript2Exe, you might want to read Distributing Ruby Applications. It's about how I build, pack and distribute my Ruby applications. Theory and practice.

(I'm working on full support of RubyGems. The handling of `require_gem` and the mangling of `$:` are implemented and all files of a gem are embedded. I've tested just a couple of gems, not all of them. If you've troubles with a specific gem, please let me know.)

2. Internals

2.1. RubyScript2Exe

RubyScript2Exe monitors the execution of your application. This is done by running your application with a special library. After your application has finished, this special library returns all information about your application to RubyScript2Exe. RubyScript2Exe then gathers all program files and requirements (`ruby.exe`, `rubyw.exe` or `ruby` (and their `so`'s, `o`'s and `dll`'s, determined recursively), `*.rb`, `*.so`, `*.o` and `*.dll` (and their `so`'s, `o`'s and `dll`'s, determined recursively)) from your own Ruby installation. All these files, your application and an extracting program are combined into one single, compressed executable. This executable can run on a bare Windows installation, a Linux installation with a libc version >= yours or a Darwin installation. Call it a "just-in-time and temporary installation of Ruby"...



2.2. EEE

EEE stands for "Environment Embedding Executable". Well, I just had to give it a name...

EEE is the little Pascal program that packs and compresses all necessary files. It had to be written in a language that could be compiled and linked into an `exe`-file. Ruby wasn't an option. I use FreePascal (1.9.8 on Windows, 1.9.8 on Linux and 1.9.8 on Darwin).

EEE has two modes: packing and unpacking. When it detects an attached archive, it jumps into unpacking mode; into packing mode otherwise.

After creating the temporary directory and unpacking all files, EEE spawns the Ruby interpreter for your *`application`*`.rb` . At that point, EEE releases control to Ruby itself. After Ruby has

finished, EEE regains control and starts cleaning up.

The difference between `eee.exe` and `eeew.exe` is the same as the difference between `ruby.exe` and `rubyw.exe`: With or without a DOS-box.

I've given EEE a page of its own, with more information then this section provides.

3. Usage

3.1. Compiling the Application

If you use the original `rubyscript2exe.rb`:

```
c:\home\erik> ruby rubyscript2exe.rbapplication.rb[w] [parameters]
or
c:\home\erik> ruby rubyscript2exe.rbapplication[/] [parameters]
```

If you installed the gem, it's:

```
c:\home\erik> rubyscript2exe application.rb[w] [parameters]
or
c:\home\erik> rubyscript2exe application[/] [parameters]
```

Parameter	Description
`--rubyscript2exe-rubyw`	Avoid the popping up of a DOS box. (It's annoying in the test period... No `puts` and `p` anymore... Only use it for distributing your application. See Logging.)
`--rubyscript2exe-ruby`	Force the popping up of a DOS box (default).
`--rubyscript2exe-nostrip`	Avoid stripping. The binaries (`ruby` and `*.so`) on Linux and Darwin are stripped by default to reduce the size of the resulting executable.
`--rubyscript2exe-strace`	Start the embedded application with strace (Linux only, for debugging only).
`--rubyscript2exe-tk`	(*experimental*) Embed not only the Ruby bindings for TK, but TK itself as well.
`--rubyscript2exe-verbose`	Verbose mode.
`--rubyscript2exe-quiet`	Quiet mode.

In case you want to compile a complete directory, the entry point of you application has to be `init.rb`. RubyScript2Exe complains if it can't find *`application`*`/init.rb`.

All parameters starting with `--rubyscript2exe-` will be deleted before the execution of *`application.rb`*.

If the extension is "rb", a DOS box will pop up. If the extension is "rbw", no DOS box will pop up. Unless it is overwritten by a parameter.

On Linux and Darwin, there's no difference between `ruby` and `rubyw`.

When using `--rubyscript2exe-tk`, it's probably a good idea to add `exit if RUBYSCRIPT2EXE.is_compiling?` (see is_compiling?) just before `Tk.mainloop`:

```
require "rubyscript2exe"
exit if RUBYSCRIPT2EXE.is_compiling?
Tk.mainloop
```

It is possible to change the icon of the generated executable manually, with a resource editor like Resource Hacker. If Resource Hacker is installed, in your %PATH% and therefor available from the current directory, and an icon file with the name *application*.ico exists in the current directory, the default icon will automatically be replaced by yours. I used Resource Hacker 3.4.0 for my tests.

3.2. Running the Application

```
c:\home\erik> application.exe [parameters]
```

Parameter	Description
--eee-list	Just list the contents of the executable. (Doesn't work in combination with rubyw.)
--eee-info	Just show the information stored in the executable. (Doesn't work in combination with rubyw.)
--eee-justextract	Just extract the original files from the executable into the current directory (no subdirectory!).

If one of these parameters is used, RubyScript2Exe does just that. It doesn't execute the application.

If none of these parameters is used, RubyScript2Exe executes the application with the given parameters. To be forward compatible, all parameters starting with --eee- will be deleted before the execution of the application.

The exit code of the executable is the same as the exit code of your application.

3.3. From Inside your Application

Module RUBYSCRIPT2EXE is available after doing a require "rubyscript2exe". This module is used in this section.

Yep, we've two files with the same name: the application rubyscript2exe.rb (big) and the library rubyscript2exe.rb (small). They're not the same. But, since the big application rubyscript2exe.rb is an RBA and includes the small library rubyscript2exe.rb, you can always do require "rubyscript2exe". It doesn't matter whether Ruby finds the big one or the small one. It should work either way. Funny stuff, those RBA's... ;]

This is an overview of the methods (or *module variables*) RUBYSCRIPT2EXE provides. They're explained in detail in the next sections.

Method	Useful at Compile-Time	Useful at Run-Time	Default
RUBYSCRIPT2EXE.dlls=	x		[]
RUBYSCRIPT2EXE.bin=	x		[]
RUBYSCRIPT2EXE.lib=	x		[]
RUBYSCRIPT2EXE.tempdir=	x		nil
RUBYSCRIPT2EXE.tk=	x		false

RUBYSCRIPT2EXE.rubyw=	x		false
RUBYSCRIPT2EXE.strip=	x		true
RUBYSCRIPT2EXE.is_compiling?	x		
RUBYSCRIPT2EXE.is_compiled?		x	
RUBYSCRIPT2EXE.appdir		x	
RUBYSCRIPT2EXE.userdir		x	
RUBYSCRIPT2EXE.exedir		x	
RUBYSCRIPT2EXE.executable		x	

a) RUBYSCRIPT2EXE.(dlls|bin|lib)=

The application itself (*application*.rb) usually doesn't need to know that it's wrapped by RubyScript2Exe. But sometimes RubyScript2Exe needs to know something about the application. Instead of introducing separate configuration files, I simply abuse *application*.rb as a configuration file...

Sometimes, you want to embed an additional DLL in the executable. That's easily done by using RUBYSCRIPT2EXE.dlls= in your application:

```
require "rubyscript2exe"
RUBYSCRIPT2EXE.dlls = ["a.dll", "b.dll", "c.dll"]
```

(You can also do this: RUBYSCRIPT2EXE.dlls << "a.dll")

At the end of the tracing of your application, the mentioned DLL's are copied from the directory in which the application was started, if they exist. The DLL's on which these DLL's depend are not copied, in contrast to the dependencies of ruby.exe and its libraries, which are resolved recursively.

(Although RubyScript2Exe knows how to handle application directories, you still have to mention your personal DLL's by hand. Yes, the DLL's are embedded twice... I want to change this in the future.)

On one location, I was not supposed to change the application for this kind of things. So I did the following trick:

```
c:\home\erik> type dlls.rb
require "rubyscript2exe"
RUBYSCRIPT2EXE.dlls = ["some.dll", "another.dll"]

c:\home\erik> ruby -r dlls rubyscript2exe.rb application.rb
```

Like RUBYSCRIPT2EXE.dlls=, you can use RUBYSCRIPT2EXE.bin= as well for EXE's and (non-library) DLL's and SO's. In fact, RUBYSCRIPT2EXE.dlls= and RUBYSCRIPT2EXE.bin= are handled exactly the same. For library files (RB's, SO's and DLL's), you can use RUBYSCRIPT2EXE.lib=.

b) RUBYSCRIPT2EXE.tempdir=

Some firewalls block outbound connections to prevent viruses and other bad programs to connect to their friends, unless the program initiating the connection is "white-listed" manually. This "white-list" is based upon the full path to the executable. RubyScript2Exe installs Ruby and the application in a temporary directory in `%TEMP%`, before starting it. This directory is something like `$HOME/.eee/eee.application.243` or `%HOME%\eee\eee.application.342`. The number part changes every time you start the application. This is not good if you want to "white-list" the program, because `ruby.exe` is started from another directory every time. To prevent this, you can set `RUBYSCRIPT2EXE.tempdir=` to the directory name that will be created in `%TEMP%`:

```
require "rubyscript2exe"
RUBYSCRIPT2EXE.tempdir = "myapplication"
```

Now RubyScript2Exe will use `$HOME/.eee/myapplication` or `%HOME%\eee\myapplication` every time the program is started. This has a drawback: A second instance of the program tries to install itself in the same directory. It fails to do so, because the directory already exists. It gets even worse when the first instance of the application dies unexpectedly and fails to cleanup its own temporary directory: You won't be able to start the application anymore, unless you remove the temporary directory manually or wait for the OS to do so.

(Use `RUBYSCRIPT2EXE.tempdir=` only when necessary! It's just a hack. Its behavior might be changed in the future. I don't know yet...)

c) RUBYSCRIPT2EXE.tk=

Embed not only the Ruby bindings for TK, but TK itself as well.

(This is considered experimental.)

d) RUBYSCRIPT2EXE.rubyw=

It's the same as compiling with `--rubyscript2exe-rubyw`.

e) RUBYSCRIPT2EXE.strip=

It's the same as compiling with `--rubyscript2exe-nostrip` (but reversed...).

f) RUBYSCRIPT2EXE.is_compil(ing|ed)?

The application is run by RubyScript2Exe on two different moments in time:

- The moment the developer creates the executable. This can be detected with `RUBYSCRIPT2EXE.is_compiling?`.
- The moment the customer runs the executable. This can be detected with `RUBYSCRIPT2EXE.is_compiled?`.

g) RUBYSCRIPT2EXE.appdir

If you want to know the full path to the directory of your (embedded) application, use `RUBYSCRIPT2EXE.appdir`. You can do this when the application is compiled, but even when it isn't yet compiled.

For example (not compiled):

```
require "rubyscript2exe"
RUBYSCRIPT2EXE.appdir             ===> C:/bin
RUBYSCRIPT2EXE.appdir("README")   ===> C:/bin/README
RUBYSCRIPT2EXE.appdir{Dir.pwd}    ===> C:/bin
```

For example (compiled):

```
require "rubyscript2exe"
RUBYSCRIPT2EXE.appdir             ===> C:/home/eee/eee.troep.exe.2/app
RUBYSCRIPT2EXE.appdir("README")   ===> C:/home/eee/eee.troep.exe.2/app/README
RUBYSCRIPT2EXE.appdir{Dir.pwd}    ===> C:/home/eee/eee.troep.exe.2/app
```

`RUBYSCRIPT2EXE.appdir` and `RUBYSCRIPT2EXE.appdir("bin")` are added to `ENV["PATH"]`.

`RUBYSCRIPT2EXE.appdir` and `RUBYSCRIPT2EXE.appdir("lib")` are added to `$:`.

h) RUBYSCRIPT2EXE.userdir

If you want to know the full path to the directory in which the user started the application, use `RUBYSCRIPT2EXE.userdir`. You can do this when the application is compiled, but even when it isn't yet compiled.

For example (not compiled or compiled):

```
require "rubyscript2exe"
RUBYSCRIPT2EXE.userdir              ===> C:/work
RUBYSCRIPT2EXE.userdir("app.cfg")   ===> C:/work/app.cfg
RUBYSCRIPT2EXE.userdir{Dir.pwd}     ===> C:/work
```

(Actually, when running the application uncompiled, this is the directory (`Dir.pwd`) in which the applications requires `rubyscript2exe.rb`, which isn't necessarily the directory in which the user started the application.)

i) RUBYSCRIPT2EXE.exedir

If you want to know the full path to the directory in which your executable resides, use `RUBYSCRIPT2EXE.exedir`. You can do this when the application is compiled, but even when it isn't yet compiled.

For example (not compiled or compiled):

```
require "rubyscript2exe"
RUBYSCRIPT2EXE.exedir              ===> C:/bin
RUBYSCRIPT2EXE.exedir("app.cfg")   ===> C:/bin/app.cfg
```

```
RUBYSCRIPT2EXE.exedir{Dir.pwd}     ===> C:/bin
```

(Actually, when running the application uncompiled, this is the directory of the main script. Literally: `File.dirname(File.expand_path($0))`.)

j) RUBYSCRIPT2EXE.executable

If you want to know the full path to the executable, use `RUBYSCRIPT2EXE.executable`. You can do this when the application is compiled, but even when it isn't yet compiled.

For example (not compiled):

```
require "rubyscript2exe"
RUBYSCRIPT2EXE.executable   ===> C:/bin/app.rb
```

For example (compiled):

```
require "rubyscript2exe"
RUBYSCRIPT2EXE.executable   ===> C:/bin/app.exe
```

(Actually, when running the application uncompiled, this is the main script. Literally: `File.expand_path($0)`.)

k) Information about EEE

In your application, you can access some information about the environment EEE sets up before spawning your application:

Constant	Set to	Replaced by
`RUBYSCRIPT2EXE::APPEXE`	Filename of the generated executable.	`RUBYSCRIPT2EXE.executable` and `RUBYSCRIPT2EXE.exedir`
`RUBYSCRIPT2EXE::EEEEXE`	`eee.exe` or `eeew.exe` or `eee_linux` or `eee_darwin`.	
`RUBYSCRIPT2EXE::TEMPDIR`	Temporary directory in which the application resides.	`RUBYSCRIPT2EXE.appdir`
`RUBYSCRIPT2EXE::PARMS`	Parameters from the command line.	
`RUBYSCRIPT2EXE::QUOTEDPARMS`	Quoted parameters from the command line.	

(Use these constants only when necessary. Don't consider them "stable"...)

3.4. Tips & Tricks

a) Just Scanning, no Running

RubyScript2Exe runs the application (in a child process) and gathers the `require`-d files. It's not necessary to run the complete application when all `require`-s are done in the first couple of statements. You might as well exit right after the `require` statements:

```
require "rubyscript2exe"
exit if RUBYSCRIPT2EXE.is_compiling?
```

Sometimes, one or more `require`-s are done later on, deep down in a library (e.g. when connecting to a database in DBI). It's not a good idea to do the above trick under this kind of circumstances. You'll miss some libraries...

b) Logging

When using `--rubyscript2exe-rubyw`, the application runs without a console. This is nice for an application with a GUI. But, although you're a good programmer, sometimes the applications simply dies. If there's no console, there's no back-trace as well. I usually add one of the following lines to the top of my application, even before the `require` statements:

```
$stdout = $stderr = File.new("/path/to/temp/application.log", "w")
or
$stdout = $stderr = File.new("/path/to/temp/application.#{Process.pid}.log", "w")
```

c) Hacking on Location

You can extract, modify and "compile" on location, if you want to.

First, extract the executable:

```
c:\home\erik> application.exe --eee-justextract
```

This creates the directory `bin` (with the files `ruby.exe`, `rubyw.exe` and `*.dll`), the directory `lib` (with the dependencies `*.rb` and `*.so`), the directory `app` (with the file *`application`*`.rb` (your script) or the application directory) and the files *`app`*`.eee` and `eee.exe` (or `eeew.exe`) in the current directory.

It's possible to run your application again with:

```
c:\home\erik> bin\ruby.exe -r .\bootstrap.rb -T1 empty.rb .\app\application.rb
```

If the application does a `Dir.chdir`, try this:

```
c:\home\erik> bin\ruby.exe -r .\bootstrap.rb -T1 empty.rb
                                          c:\full\path\to\app\application.rb
```

After hacking *`app`*`.eee`, if necessary, you can "compile" your application again with:

```
c:\home\erik> eee.exe app.eee newapplication.exe
or
c:\home\erik> eeew.exe app.eee newapplication.exe
```

On Linux, it's pretty much the same.

```
$ ./application_linux --eee-justextract

$ export PATH=./bin:$PATH
$ export LD_LIBRARY_PATH=./bin:$LD_LIBRARY_PATH
$ chmod +x ./bin/*
$ ./bin/ruby -r ./bootstrap.rb -T1 empty.rb ./app/application.rb

$ ./eee_linux app.eee newapplication_linux
```

4. Examples

4.1. Distributions

I ran RubyScript2Exe with four different Ruby distributions (Ruby 1.8.1) on Windows and two versions of Ruby (1.6.7 and 1.8.2) on Linux:

Distribution	Size (bytes)
Cygwin	1287227
RubyInstaller	641840
MinGW	428898
MSWin32	467110
Linux, Ruby 1.6.7	551858
Linux, Ruby 1.8.2	574015

The details can be found here.

The test script was nothing more than a little Hello World thing (And the `require "rbconfig"` was just an extra test item...):

```
require "rbconfig"

puts "Hello World!"
```

5. License

5.1. License of RubyScript2Exe

RubyScript2Exe, Copyright (C) 2003 Erik Veenstra <rubyscript2exe@erikveen.dds.nl>

This program is free software; you can redistribute it and/or modify it under the terms of the *GNU General Public License (GPL), version 2*, as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, **but without any warranty**; without even the implied warranty of **merchantability** or **fitness for a particular purpose**. See the *GNU General Public License (GPL)* for more details.

You should have received a copy of the *GNU General Public License (GPL)* along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

The full text of the license can be found here.

5.2. License of your Application

Whatever...

6. Download

Current version is 0.5.3 (29.05.2007). It's a stable release.

Tested on:

- Red Hat Linux 8.0 with Ruby 1.6.7
- Red Hat Linux 8.0 with Ruby 1.8.1
- Red Hat Linux 8.0 with Ruby 1.8.2
- Windows 95 with Ruby 1.8
- Windows 98 with Ruby 1.6 (Very slow!)
- Windows 98 with Ruby 1.8
- Windows 2000 with Ruby 1.8
- Windows 2000 with Ruby 1.8 (Cygwin)
- Windows XP with Ruby 1.8
- Windows XP with Ruby 1.8 (Cygwin)

You only need rubyscript2exe.rb . It's the current version, packed as an RBA (Ruby Archive, built by Tar2RubyScript) and works on both Windows and Linux. You can download rubyscript2exe.tar.gz if you want to play with the internals of RubyScript2Exe. RubyScript2Exe is available as rubyscript2exe.gem as well.

Send me reports of all bugs and glitches you find. Propositions for enhancements are welcome, too. This helps *us* to make *our* software better.

A change log and older versions can be found here. A generated log file can be found here.

RubyScript2Exe is available on SourceForge.net and on RubyForge .

6.1. Mac OS X (Darwin)

I included (experimental) support for Darwin. The Ruby code in the above mentioned packages is able to handle Darwin, but the packages don't include EEE for Darwin. (They would be too big...) For now, you have to compile it yourself:

1. Get `eee.pas` from the archive.
2. Download the compiler.
3. Compile (`fpc -Xs -B eee.pas`).
4. Rename `eee` to `eee_darwin`.

(I've put a precompiled `eee_darwin` on my site, but it may be newer than (and therefor incompatible with) the released Ruby code.)

RubyScript2Exe searches for `eee_darwin` (or `eee_linux` or `eee.exe` or `eeew.exe`) in 3 locations:

1. In `rubyscript2exe.rb` (or `rubyscript2exe/` when using `rubyscript2exe.tar.gz`).
2. In the directory in which `rubyscript2exe.rb` is located.

3. In the current directory.

This means that you can simply put `eee_darwin` in the same directory as `rubyscript2exe.rb` (location 2) or in the current directory (location 3).

If you want to repackage RubyScript2Exe (location 1) with an embedded `eee_darwin`, do this:

1. Extract rubyscript2exe.tar.gz, or extract rubyscript2exe.rb (`ruby rubyscript2exe.rb --tar2rubyscript-justextract`)
2. Copy `eee_darwin` to `rubyscript2exe/`.
3. Recreate rubyscript2exe.rb (`ruby tar2rubyscript.rb rubyscript2exe/`) (optional)

7. Known Issues

- Don't use long application names (as in *thisisalongnameofanapplication*.rb). Long application names result in non-working executables. Somehow `gzread`, used to read a block from the archive, returns -2 if `"#{eeedir}\\eee.#{application}.exe.2\\eee.gz".length >= 80`.
- RubyScript2Exe is tested with RubyGems 0.9.0. It might be necessary to add `include "fileutils"` in your application.
- If you someday run into trouble when trying to generate RDOC documentation from within the generated executable, have a look at Zoran Lazarevic's page.
- Because of the way Ruby is started from within RubyScript2Exe on Linux, `$stdin` doesn't work anymore. On Windows, it works. **FIXED IN 0.3.4!**
- My latest change in RubyScript2Exe was to ignore RUBYOPT when running the generated executable. This means that I do generate the fake rubygems.rb and do embed it into the executable, but it isn't necessarily required by anything in the application or in the environment. So, you have to do a `require "rubygems"` **explicitly** in your application, when using one or more gems. **FIXED IN 0.3.3!**
- Running the generated executable from within a non-SH-compatible shell (e.g. TCSH), is currently not possible. **FIXED IN 0.3.3!**
- There is a problem when running the generated executable in an environment in which the environment variable `RUBYOPT` is set. (The One-Click Ruby Installer (`ruby182-14.exe`) does this.) The embedded `ruby.exe` encounters `RUBYOPT` and tries to load the mentioned libraries (e.g. `ubygems`) which it obviously can't find, because the search paths are pointing to the embedded environment and not to the original environment. So, **creating** an executable in an environment in which `RUBYOPT` is set is not a problem, whereas **running** the generated executable in such an environment is a problem. The workaround is to open a DOS-box, do a `set RUBYOPT=` and run the executable from within the DOS-box. **FIXED IN 0.3.2!**