Tar2RubyScript - A Tool for Distributing Ruby Applications

Table of Contents

Tar2RubyScript **1**
A Tool for Distributing Ruby Applications 1

1. Introduction **2**

2. Internals **3**
2.1. Creating the RBA 3
a) General 3
b) tar2rubyscript.bat and tar2rubyscript.sh 3
2.2. Executing the RBA 3

3. Usage **5**
3.1. Tar2RubyScript 5
a) General Invocation 5
b) The TAR Archive Variant 5
c) The Directory Variant 6
3.2. init.rb 6
3.3. RBA (An Application) 7
3.4. RBA (A Library) 7

4. Examples **9**
4.1. Usage of Tar2RubyScript 9
a) The TAR Archive Variant 9
b) The Directory Variant 9
c) A Library (Without init.rb) 9
d) A Library (With init.rb) 10
4.2. tar2rubyscript.sh and tar2rubyscript.bat 10
4.3. oldlocation and newlocation 10
4.4. Combination of Tar2RubyScript and RubyScript2Exe 11

5. License **12**
5.1. License of Tar2RubyScript 12
5.2. License of your Application 12

6. Download **13**

7. Known Issues **14**

8. Credits **15**

Notes **16**

Tar2RubyScript

A Tool for Distributing Ruby Applications

Fri May 25 15:06:29 UTC 2007

Erik Veenstra <tar2rubyscript@erikveen.dds.nl>

1. Introduction

Tar2RubyScript transforms a directory tree, containing your application, into **one single Ruby script**, along with some code to handle this archive. This script can be distributed to our friends. When they've installed Ruby, they just have to double click on it and your application is up and running!

So, it's a way of executing your application, not of installing it. You might think of it as the Ruby version of Java's JAR... Let's call it an RBA (Ruby Archive).

"It's Ruby's JAR..."

Like packing related application files into one RBA application, you could as well pack related library files into one RBA library. Now you don't need to install the compound library in the traditional way before using it. Just require the RBA.

Because the RBA is pure Ruby and no other programs or libraries are needed, it's easy to distribute it to friends. They don't have to install anything but Ruby itself.

Unlike the JAR-people, we don't need a new extension for RBA's. A JAR isn't a Java Class, it contains a Java class; an RBA both is and contains a Ruby script. It's also easier to change the format of an RBA in the future, because the algorithm to handle the RBA comes with it at a cost in bytes of less then 10K. Another difference between the two is the entry point: JAR does something with a manifest; RBA just loads `init.rb` . And, well, they compress, we don't.

If you like Tar2RubyScript, you might want to read Distributing Ruby Applications. It's about how I build, pack and distribute my Ruby applications. Theory and practice.

The combination of Tar2RubyScript and RubyScript2Exe is of special interest: A complete Ruby application can be distributed as one executable:

- Tar2RubyScript creates a standalone Ruby script (or RBA, Ruby Archive) of the application and its directory. This RBA can run on "the Ruby platform". This means that Ruby itself must be installed on the targeted system.
- RubyScript2Exe avoids this dependency by compiling a rubyscript (in casu the RBA), the Ruby interpreter and the Ruby runtime environment into one executable.

This combination isn't necessary. Each application can be used without the other.

2. Internals

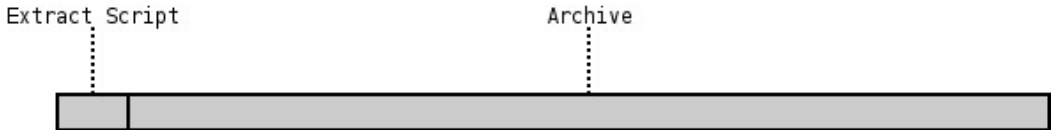
2.1. Creating the RBA

a) General

Tar2RubyScript generates a base64 encoded string from a TAR archive or directory, which contains a Ruby application, and attaches that string to a Ruby extraction script. The result is just one file, a pure Ruby script, containing multiple files, stored as comments.

Tar2RubyScript contains two scripts: `init.rb`, which is the script that is run when building the RBA, and `tarrubyscript.rb`, which is the script that is run when unpacking/running the RBA. In fact, `tarrubyscript.rb` **is** the RBA, without the archive. These scripts run on two distinct moments in time.

The external program `tar` is still needed for generating the TAR archive internally. (Once again: Not for extracting/using it.) I assume all Linux and Unix systems have one. I embedded `tar.exe`, because only a few Windows systems have it.



b) tar2rubyscript.bat and tar2rubyscript.sh

When creating the RBA, the archive/directory is extracted/copied to a temporary directory. If the archive/directory contains `tar2rubyscript.bat` (Windows) or `tar2rubyscript.sh` (Linux, Cygwin (Posix in general?)), that script will be executed in the temporary directory. Finally, the temporary directory is packed to a new TAR archive which becomes part of the RBA. Thus, the original TAR archive isn't necessarily the same as the one which is part of the RBA. This can be avoided with `--tar2rubyscript-preserve`. When using `--tar2rubyscript-preserve` in combination with a TAR archive, no external TAR commands are run. No unpacking, no packing. This might be necessary on platforms with deviant behavior (e.g. TAR on Solaris). Using `--tar2rubyscript-preserve` in combination with a directory is pointless.

2.2. Executing the RBA

When the RBA is executed, it splits up into two parts (the script and the string), decodes the base64 encoded string to a temporary TAR archive, unpacks the archive to a temporary directory, jumps to that directory, eventually jumps to an existing subdirectory, and finally loads `init.rb`. Et voila, your application is up and running. When the application has ended, the RBA deletes the temporary directory recursively.

When unpacking the archive, everything but paths, filenames and contents is disregarded. No permissions, no owners, no groups. However, hard links and soft links are supported (not on Windows). Note that both hard and soft links make your RBA platform dependent.

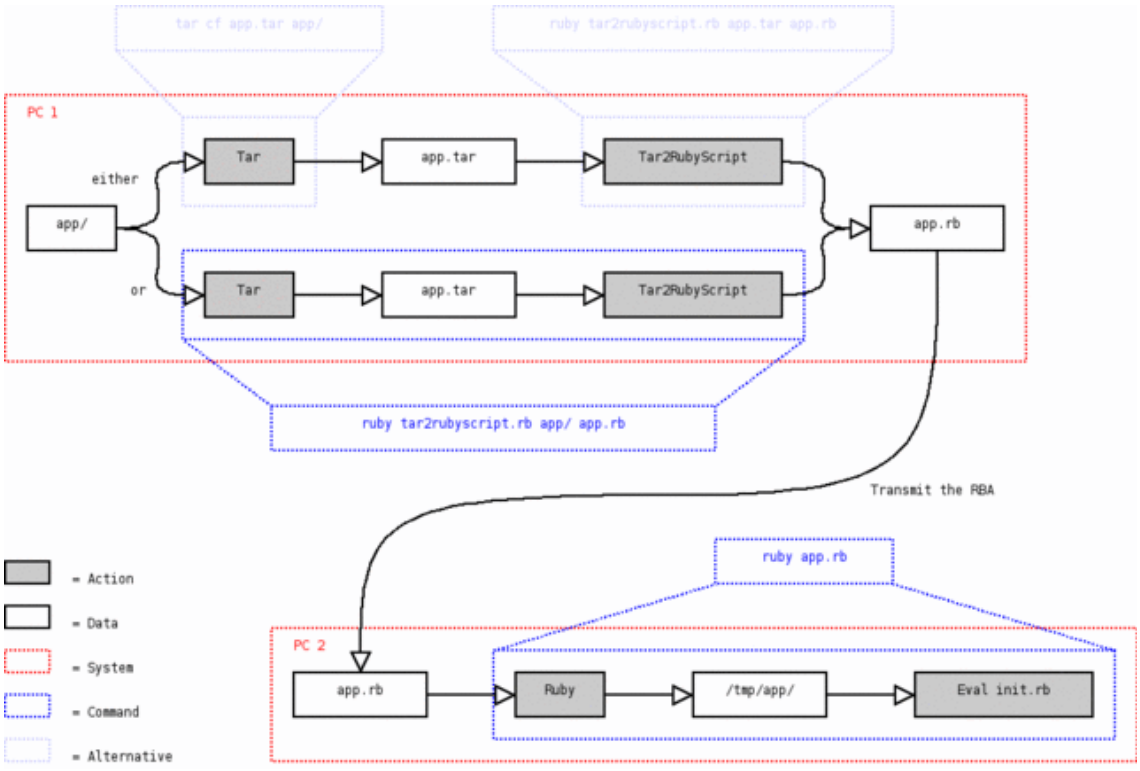
2. Internals

(Use these links with care. It's not thoroughly tested...)

Because of the cleaning of temporary files (done by the OS), all files in the temporary directory are touched every hour. This prevents the deletion of files of long running jobs. This is done by a background thread that keeps visiting all temporary files in the temporary directory and its subdirectories.

`ENV["PATH"]` is extended with both `oldlocation` and `newlocation`, just before loading `init.rb`. The OS might need this to find shared objects (e.g. DLL's).

3. Usage



3.1. Tar2RubyScript

a) General Invocation

If you use the original `tar2rubyscript.rb`:

```
c:\home\erik> ruby tar2rubyscript.rb ...
```

If you installed the gem, it's:

```
c:\home\erik> tar2rubyscript ...
```

b) The TAR Archive Variant

Create a TAR archive, with a root that contains at least the file `init.rb` or a root that contains just one directory which contains at least the file `init.rb`. If the root of the archive contains only one directory, a `cd` to that directory will be done when executing the RBA.

Create the RBA:

```
c:\home\erik> tar cf application.tar application[/]
c:\home\erik> ruby tar2rubyscript.rb application.tar \
                                    application.rb [license.txt]
```

If the TAR archive also contains `tar2rubyscript.bat` (Windows) or `tar2rubyscript.sh` (Linux, Cygwin (Posix in general?)), that script will be executed "in the archive". Changes in the archive aren't permanent.

If *`application`*`.rb` is not provided or if it equals to "-", it will be derived from *`application`*`.tar` .

If a license is provided, it will be put at the beginning of the RBA.

c) The Directory Variant

Create a directory, which contains at least the file `init.rb`.

Create the RBA:

```
c:\home\erik> ruby tar2rubyscript.rb application[/] \
                                    application.rb [license.txt]]
```

If the TAR archive also contains `tar2rubyscript.bat` (Windows) or `tar2rubyscript.sh` (Linux, Cygwin (Posix in general?)), that script will be executed in a copy of the directory, just before the internal TAR archive is being created.

(Changes in the directory aren't permanent anymore!)

If *`application`*`.rb` is not provided or if it equals to "-", it will be derived from *`application/`* .

If a license is provided, it will be put at the beginning of the RBA.

3.2. init.rb

Because `init.rb` executes in the temporary directory, all references to files relative to the original directory have to be escaped [1] by `oldlocation`. For example, instead of...

```
file = ARGV.shift
```

...you have to use:

```
file = oldlocation(ARGV.shift)
```

If you want to execute a block of code in the original directory, you can use `oldlocation` as well:

```
oldlocation do
  Dir.new('.').each do |f|
    # ...
  end
end
```

Put this at the beginning of the script to let it run in the original directory (see the examples):

```
Dir.chdir oldlocation
```

Now you have to escape the files in the temporary directory:

```
file = newlocation("...")
or
newlocation{...}
```

I created `oldandnewlocation.rb` that can be "`require`"d in `init.rb`, so the application can run without being wrapped by Tar2RubyScript. This script doesn't do anything but defining `oldlocation` and `newlocation`.

`TAR2RUBYSCRIPT` is set to true before loading `init.rb`, so it's possible to detect whether the application is run as an RBA or not:

```
puts "Running as an RBA." if defined?(TAR2RUBYSCRIPT)
```

3.3. RBA (An Application)

`c:\home\erik> ruby` *`application`*`.rb [`*`parameters`*`]`

Parameter	Description
--tar2rubyscript-list	Just list the contents of the RBA.
--tar2rubyscript-justextract	Just extract the RBA, but not execute it.
--tar2rubyscript-totar	Just convert the RBA (back) into a TAR file.

If one of these parameters is used, Tar2RubyScript does just that. It doesn't execute `init.rb`.

If none of these parameters is used, Tar2RubyScript executes `init.rb` with the given parameters. To be forward compatible, all parameters starting with `--tar2rubyscript-` will be deleted before the execution of `init.rb` .

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

3.4. RBA (A Library)

Packing a library into an RBA is basically the same as packing an application. It's just that you don't need the `init.rb`.

Using it is different from an application. Suppose you packed a directory with the file `lib_a.rb` and `lib_b.rb` into the RBA `testlib.rb`. An application, which is going to use both libraries, needs to `require "testlib"`, before `require`-ing the two libraries:

```
require "testlib"
require "lib_a"
require "lib_b"
```

I usually put the last two lines in the `init.rb` of the RBA. That will reduce the `require`-ing in the application to just one line:

```
require "testlib"
```

It's possible to use multiple libraries in one library RBA and multiple library RBA's in your application, which can be an RBA as well. `newlocation` escapes the files of the last RBA, either library or application. It's not possible (yet) to access the files in other RBA's.

4. Examples

4.1. Usage of Tar2RubyScript

a) The TAR Archive Variant

```
$ rm -rf demo demo.tar demo.rb
$
$ mkdir demo
$
$ echo 'puts File.new("demo1.txt").readlines' >  demo/init.rb
$ echo 'puts File.new("demo2.txt").readlines' >> demo/init.rb
$ echo 'Hello                                 ' >  demo/demo1.txt
$ echo 'World!                                ' >  demo/demo2.txt
$
$ tar cf demo.tar demo/
$
$ ruby tar2rubyscript.rb demo.tar          # Results in demo.rb
$
$ ruby demo.rb
Hello
World!
```

b) The Directory Variant

```
$ rm -rf demo demo.tar demo.rb
$
$ mkdir demo
$
$ echo 'puts File.new("demo1.txt").readlines' >  demo/init.rb
$ echo 'puts File.new("demo2.txt").readlines' >> demo/init.rb
$ echo 'Hello                                 ' >  demo/demo1.txt
$ echo 'World!                                ' >  demo/demo2.txt
$
$ ruby tar2rubyscript.rb demo/          # Results in demo.rb
$
$ ruby demo.rb
Hello
World!
```

c) A Library (Without init.rb)

```
$ rm -rf demo demo.tar demo.rb
$
$ mkdir demo
$
$ echo 'puts "Hello"                          ' >  demo/lib_a.rb
$ echo 'puts "World!"                         ' >  demo/lib_b.rb
$
$ ruby tar2rubyscript.rb demo/          # Results in demo.rb
$
$ ruby -e 'require "demo" ; require "lib_a" ; require "lib_b"'
Hello
World!
```

d) A Library (With init.rb)

```
$ rm -rf demo demo.tar demo.rb
$
$ mkdir demo
$
$ echo 'require "lib_a.rb"                ' >  demo/init.rb
$ echo 'require "lib_b.rb"                ' >> demo/init.rb
$ echo 'puts "Hello"                      ' >  demo/lib_a.rb
$ echo 'puts "World!"                     ' >  demo/lib_b.rb
$
$ ruby tar2rubyscript.rb demo/         # Results in demo.rb
$
$ ruby -e 'require "demo"'
Hello
World!
```

4.2. tar2rubyscript.sh and tar2rubyscript.bat

```
$ rm -rf demo demo.tar demo.rb
$
$ mkdir demo
$
$ echo 'puts File.new("demo1.txt").readlines' >  demo/init.rb
$ echo 'puts File.new("demo2.txt").readlines' >> demo/init.rb
$ echo 'Hello                             ' >  demo/demo1.txt
$ echo 'echo World! > demo2.txt           ' >  demo/tar2rubyscript.sh
$
$ ruby tar2rubyscript.rb demo/         # Results in demo.rb
". ./tar2rubyscript.sh"
$
$ ruby demo.rb
Hello
World!
```

4.3. oldlocation and newlocation

```
$ rm -rf demo demo.tar demo.rb
$
$ mkdir demo
$
$ echo 'p [1, oldlocation("file.ext")]    ' >  demo/init.rb
$ echo 'p [2, newlocation("file.ext")]    ' >> demo/init.rb
$ echo '                                  ' >> demo/init.rb
$ echo '      p [3, Dir.pwd]              ' >> demo/init.rb
$ echo 'oldlocation do                    ' >> demo/init.rb
$ echo '      p [4, Dir.pwd]              ' >> demo/init.rb
$ echo '  newlocation do                   ' >> demo/init.rb
$ echo '      p [5, Dir.pwd]              ' >> demo/init.rb
$ echo '  end                             ' >> demo/init.rb
$ echo '      p [6, Dir.pwd]              ' >> demo/init.rb
$ echo 'end                               ' >> demo/init.rb
$ echo '      p [7, Dir.pwd]              ' >> demo/init.rb
$
$ ruby tar2rubyscript.rb demo/         # Results in demo.rb
$
$ ruby demo.rb
```

```
[1, "/home/erik/file.ext"]
[2, "/tmp/tar2rubyscript.d.597/demo/file.ext"]
[3, "/tmp/tar2rubyscript.d.597/demo"]
[4, "/home/erik"]
[5, "/tmp/tar2rubyscript.d.597/demo"]
[6, "/home/erik"]
[7, "/tmp/tar2rubyscript.d.597/demo"]
```

4.4. Combination of Tar2RubyScript and RubyScript2Exe

Create a directory *`application/`* which contains at least `init.rb` .

```
$ ruby tar2rubyscript.rb application/
$ ruby rubyscript2exe.rb application.rb
```

5. License

5.1. License of Tar2RubyScript

5.2. License of your Application

Whatever...

6. Download

Current version is 0.4.8 (08.03.2006). 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
- Solaris 2.8 with Ruby 1.8.0 (Only the TAR archive variant works. The directory variant doesn't.)
- Windows 95 with Ruby 1.8
- Windows 98 with Ruby 1.6 (Very slow!)
- Windows 98 with Ruby 1.8
- Windows NT 4.0 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 tar2rubyscript.rb . It's the current version, built by Tar2RubyScript itself. You can download tar2rubyscript.tar.gz if you want to play with the internals of Tar2RubyScript. Tar2RubyScript is available as tar2rubyscript.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.

Tar2RubyScript is available on SourceForge.net and on RubyForge .

7. Known Issues

- You might have to add `$stdout.sync = $stderr.sync = true` at the beginning of your script when you use `print`. Somehow, the output isn't always displayed as expected.
- On Darwin, you might run into a `Too many open files - getcwd (Errno::EMFILE)`. I've no solution for this. As far as I know, this is not directly related to Tar2Rubyscript. It's an OS kind of thing. Run ulimit -n to show the maximum number of open files. It is 256 on Mac OS X. Run ulimit -n3000 (for example) to fix it.
- Windows (at least Windows 98; Windows 2000 looks alright) doesn't wait for a child to complete, before continuing in the parent. This means that `tar2rubyscript.bat` hasn't finished copying (it's usually used to copy files into the application), before Tar2RubyScript starts packing.
- If the original directory contains read-only files, these can't be overwritten by a second file (e.g. `require "abc"; require "ABC"` : the second file will overwrite the first one, on Windows).
- Old Windows shells don't do any command line expansion. That means that TAR receives `*` as an parameter and handles it properly. Newer Windows shells do some kind of command line expansion. The `*` is replaced by `""` (Windows wants `*.*`, although there is no `.` in *app/*...), so TAR gets no parameters and complaints (`Cowardly refusing to create an empty archive`). **FIXED IN 0.4.5!**

8. Credits

- Thomas Hurst, for giving us some Ruby code for handling TAR archives.

Notes

[1] *I once heard: "I don't want to change my scripts to be compatible with your tool!". And I agree. I've had some thoughts in that direction before I introduced oldlocation and newlocation. But that ideology is less important than a solid solution for deciding which files are to be loaded from the temporary directory (help files, images, licenses, default configurations, etc.) and which files are to be loaded from the user directory (databases, user configurations, etc.). This decision can't be made by Tar2RubyScript. Somehow, the programmer has to make this decision. And it's not only about files, but also about blocks of code. (e.g. Dir.new('.')) Sometimes, you want to open the user directory; sometimes, you want to open the temporary directory.) It's up to the programmer. That's where oldlocation and newlocation popped up.*