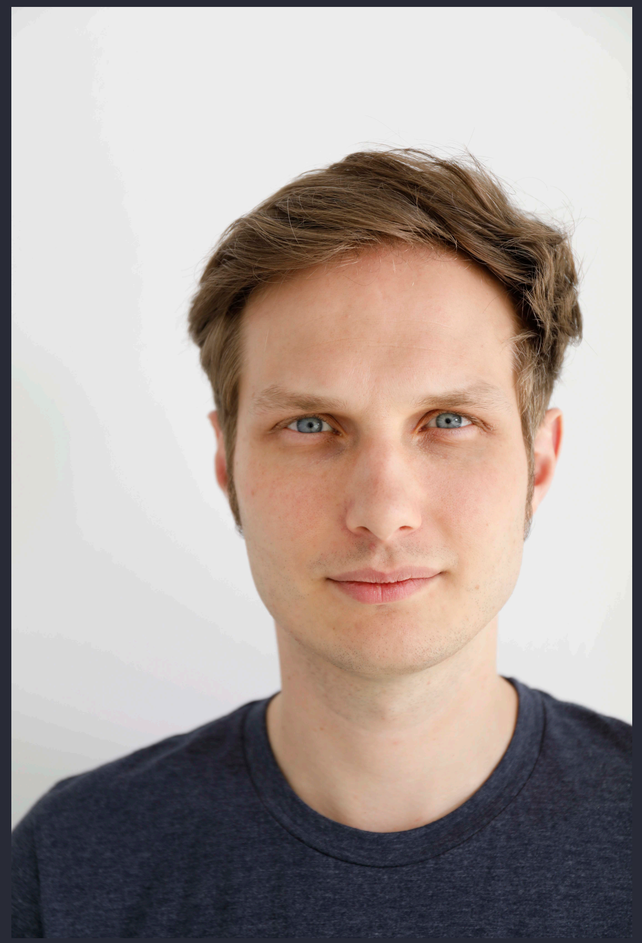
The Lost Art of Single File Ruby Programs

Christian Bäuerlein - @fabrik42

Hi!

Hi! My name is Christian Bäuerlein



- Living in Frankfurt am Main
- Leading the Technology & Engineering stream at ioki GmbH - a Deutsche Bahn company
- We build on-demand mobility platforms to enable public transport providers to roll their own DRT and autonomous services.
- christianbaeuerlein.com
- @fabrik42
- github.com/fabrik42

The Lost Art of Single File Ruby Programs

A loose collection of Ruby fun facts and examples
to organize your code in a single file.

Let's have some fun with Ruby!

A little Ruby history

Ruby is a better Perl

Why the name *Ruby*?

> Influenced by Perl, Matz wanted to use a jewel name for his new language, so he named Ruby after a colleague's birthstone.

Source: The Ruby Language FAQ

Perl’s legacy

Ruby took a lot of things from Perl.

Today we will learn about:

- Keywords
- Command line flags

Are you ready? 

Code and tests in one file

Ruby’s pre-defined variables

There is `$0`.

> Contains the name of the file containing the Ruby script being executed.

Source: Pre-defined variables and constants

See also `$PROGRAM_NAME`

Ruby’s magic keywords

There is `__FILE__`.

> The path to the current file.

Source: ruby-doc.org Keywords

The source file

```
$ cat greeter.rb

def greet(name)
  "Hello #{name}!"
end

# this will only run if the script was called directly
# not loaded or required
if __FILE__ == $0
  require "test/unit/assertions"
  include Test::Unit::Assertions

  assert_equal 'Hello Ruby', greet('Ruby'), “returns 'Hello Ruby!'"
end
```

When called directly

```
$ ruby greeter.rb

 returns 'Hello Ruby!'. (Test::Unit::AssertionFailedError)
<"Hello Ruby"> expected but was
<"Hello Ruby!">.

diff:
- Hello Ruby
+ Hello Ruby!
?
```

When required from another file

```
$ cat code_and_test_usage.rb

require ‘./greeter.rb'

puts greet "Christian"


$ ruby code_and_test_usage.rb

Hello Christian!
```

The `__END__` and `DATA` keywords

Let’s start with Perldata

Perl has two special literals:

`__END__`
Indicates the logical end of the script before the actual end of file.

`__DATA__`
A filehandle that points to everything that comes after **`__END__`**.

Source: perldata - perldoc.perl.org

The `__END__` and `DATA` keywords in Ruby

Denotes the end of the regular source code section of a program file.
Lines below`__END__` will not be executed.

Those lines will be available via the special filehandle **`DATA`**.

Source: Ruby-doc - Class: Object

Simple Example: A valid Ruby file

```
DATA.each_line do |line|
  puts line
end

__END__
Cats
Dogs
Mice
```

ERB template and code in one file

```
require 'erb'

time = Time.now
renderer = ERB.new(DATA.read)
puts renderer.result()

__END__
The current time is <%= time %>.
```

A web server in one file

Sinatra has taken the stage

```
require 'rubygems'
require 'sinatra'

get '/' do
 'Hello World'
end
```

Sinatra Templates: Uncool way

```
template :index do
  '%div.title Hello World!'
end
```

As documented in the 0.6 README.rdoc there was also a cool way to do it.

Sinatra Templates: Cool way

```
get '/' do
  haml :index
end

use_in_file_templates!

__END__
@@ layout
%body
  = yield

@@ index
%h1 Hello world!!!!!
```

Two templates, one file?

```
File.read(caller.first.split(":").first).split("__END__", 2).last
```

Source: Mixing code and data in Ruby

Bundler Inline

Bundler Fun Fact

- You don't need a **`Gemfile`** to use **`bundler`**!
- Useful for *Single File Programs™*
- Useful for scripts in your **`/utils`** folder that you only use once a year

Source: How to use Bundler in a single-file Ruby script

Example

```ruby
require 'bundler/inline'

gemfile do
  source 'https://rubygems.org'
  gem 'httparty'
end

puts HTTParty.get('https://www.boredapi.com/api/activity')
```

What happens: checks dependencies, installs dependencies, runs code.

No Gemfile.lock will be written either.

Example: Inline MiniTest suite

```ruby
require 'bundler/inline'

gemfile do
  source 'https://rubygems.org'
  gem 'minitest', require: false
end

require 'minitest/autorun'

class MyTest < Minitest::Test
  def test_should_be_true
    assert_equal true, true
  end
end
```

Advanced Example: Download iCal to org

- Install Dependencies
- Do stuff (download calendar events)
- Render to ERb template (in org-Mode format)

Source: ical_to_org.rb

The `BEGIN` and `END` keywords

Yes, this is taken from Perl as well

`BEGIN` defines a block that is run before any other code in the current file.

Similarly **`END`** defines a block that is run after any other code.

Source: Ruby Docs Miscellaneous Syntax

See also `Kernel#at_exit`

Example

```
END { puts 3 }
BEGIN { puts 1 }
puts 2
```

```
$ ruby begin.rb

1
2
3
```

Introducing LRuby

Logging Ruby

The Ruby alias for the forgetful scripter

Only Feature:

No more scrolling through your terminal...

Logs the output of a script to the script itself!

Let’s try this out!

```
$ cat log_results/hello_world.rb
```

```
$ ruby log_results/hello_world.rb
```

Introducing LRuby

```
$ lruby log_results/hello_world.rb

$ cat log_results/hello_world.rb
```

How does it work?

```
$ which lruby

lruby: aliased to
ruby -r ~/single-file-ruby-programs/lruby.rb
```

Require files via command line flag

```
ruby -r [filename]
```

> Causes Ruby to load the file using `require`.

Source: Ruby Docs Command line Options

LRuby code

```ruby
BEGIN {
  $stdout = StringIO.new
}

END {
  output = $stdout.string

  end_marker = '__END__'
  code, data = File.read($0).split(end_marker)

  time = Time.now.strftime('%Y-%m-%dT%H:%M:%S%z')
  headline = "----- [#{time}] RESULTS -----"
  new_data = [data, headline, output].join("\n")
  File.write($0, [code, new_data].join("#{end_marker}"))
  STDOUT.puts output
}
```

Finally: Fire and forget!

The Garbage flag

Aaaaand back to Perl

`perl -x`

> Leading garbage will be discarded until the first line
>
> that starts with `#!` and contains the string `"perl"`.

Source: perlrun - perldoc.perl.org

Aaaaand back to Perl

`perl -x`

> Leading garbage will be discarded until the first line
>
> that starts with `#!` and contains the string `"perl"`.

Source: perlrun - perldoc.perl.org

But… why?

`perl -x`

> Tells Perl that the program is embedded in a larger chunk of unrelated text, such as in a mail message.

Source: perlrun - perldoc.perl.org

And in Ruby?

`ruby -x`

Tells Ruby that the script is embedded in a message.

Leading garbage will be discarded until the first line

that starts with **`"#!"`** and contains string **`"ruby"`**.

Source: Ruby Docs Command line Options

And in Ruby?

`ruby -x`

Tells Ruby that the script is embedded in a message.

Leading garbage will be discarded until the first line

that starts with **`"#!"`** and contains string **`"ruby"`**.

Source: Ruby Docs Command line Options

Example: A valid Ruby program

```
Hello dear friend,
this is a mail message. Please execute it with your ruby interpreter.

Thanks, a random stranger
#! hahaha this is ruby now
puts "Hello World"
```

```
$ ruby -x email.eml
Hello World
```

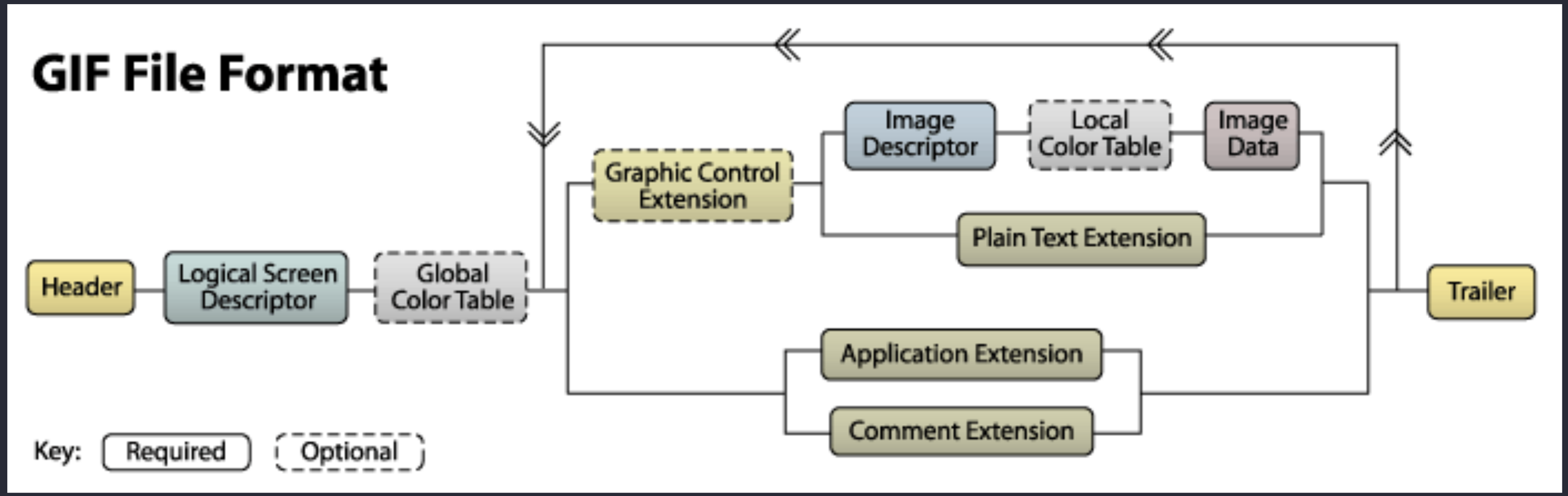
A self-animating GIF

A self-animating GIF?

This is not an animated GIF,
but a GIF that animates itself.

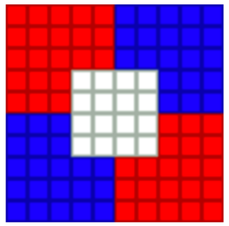
Let’s talk about GIFs

A GIF file consists of blocks



Example

Enlarged



(100x100)

Bytes

```
47 49 46 38 39 61 0A 00 0A 00 91 00 00 FF FF FF FF 00 00 00 00 FF 00 00
00 21 F9 04 00 00 00 00 00 2C 00 00 00 00 0A 00 0A 00 00 02 16 8C 2D 99
87 2A 1C DC 33 A0 02 75 EC 95 FA A8 DE 60 8C 04 91 4C 01 00 3B
```

Terminator Byte

The trailer block indicates when you've reached the end of the file.

It is always a byte with a value of `3B`.

btw: The hexadecimal value `3B` is `59` in decimal.

The ascii value `59` is a semicolon.

Source: What's in a GIF

What we learned so far

- GIFs are nice
- GIFs always end with the same terminator byte
- Ruby is nice
- Ruby can start with a defined start line
- Nice.

A self-animating GIF!

Demo time!

Let's check out the `rbgif.gif` source code together!

What will happen now

- A loop in my shell will keep calling the ruby script (gif file)
- The file will rewrite itself (the upper part aka the gif part)
- We will watch the progress in the browser

DEMO

Summary

Summary: Single File Ruby Programs

What we learned

- Code & Tests
- Dependencies & Code
- Data & Code
- Code & Data
- Code & Output
- **Try it out for fun and profit!**

Thanks!

Christian Bäuerlein - @fabrik42

Single File Ruby Programs

- Code and Slides at https://github.com/fabrik42/single-file-ruby-programs
- LRuby repository https://github.com/fabrik42/lruby