Ruby - Feature #20196

Proposal: Binary data literal

01/20/2024 02:59 AM - ziggythehamster (Keith Gable)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I sometimes find myself needing to write some bytes in a Ruby string literal, and this experience leaves a lot to be desired:

- Bare strings don't work (potential for encoding corruption unless you remember to .force_encoding and never copy-paste just the literal into somewhere else) and are not particularly pleasant given all of the backslashes
- Wrapping this in String.new("\x89PNG\r\n\x1A\n\x00\x00\x00\rIHDR\x00\x00\x00\x00\x00\x00\b\x06\x00\x00\x00\xE2\x98w8\x00\x000%IDAT", encoding: 'BINARY') is better, but many tools explode with this because they expect all strings to be valid UTF-8 even if they're an argument to String.new, and it still doesn't have the "beauty" one might expect from Ruby (also it's not frozen unless you also freeze it)
- ["9805e474d0a0a1a0000000d0948444250000000600000006806000000002e897783000003529444144 5"].pack("h*") parses in all tools and is less harsh to look at, but if you're writing binary data, you probably want to annotate it

Here's my basic syntax proposal:

```
%b[
  89504e470d0a1a0a # PNG header
  0000000d         # Length = 13 bytes
  49484452         # IHDR chunk
  00000060         # Width = 96px
  00000060         # Height = 96px
  08 06            # 8bpp RGBA
  00 00 00         # deflate / no filter / non-interlaced
]

# => "\x89PNG\r\n\x1A\n\x00\x00\x00\rIHDR\x00\x00\x00`\x00\x00\x00`\b\x06\x00\x00\x00"
```

More formally:

- To match the nibble ordering of a regular string escape, the hex characters are high nibble first (the same as the H unpack character).
- It follows the same rules as other percent literals, and I am flexible on what character is used. I chose b because h could be confusing paired with the h/H unpack characters and the inverted meaning.
- We could say that high-nibble-first is capitalized and the lower-case version is low-nibble-first, but I imagine most people will want high-nibble-first. We could also say that %b[] returns a String but %B[] returns an IO::Buffer, which has greater utility than having the capability of writing low-nibble-first literals
- Whitespace is ignored
- Comments are allowed
- The encoding is always Encoding::BINARY
- The resulting string is always frozen (and if %B[] means buffer then that is read-only as well)
- a-f can also be written A-F

Things to consider:

- Interpolation could be allowed like normal strings
- Embedding strings could be allowed (example below)
- ? literals (characters) should be handled identically to how other kinds of strings are embedded if that is allowed
- If interpolation is allowed and you interpolate a number, this should either interpolate .to_s as you would expect in a string or raise an error, because there is no unsurprising way to take a number and convert it to one or more bytes
- Strings encoded as Encoding::BINARY could have their .inspect output use this literal
- When dealing with bitmasks, it's often convenient to write them out in binary instead of hex so the on bits are easier to identify, but there is no syntax for that here that I am fond of... but someone might have an idea. I thought about .00001111 or !00001111 with mandatory whitespace before resuming hex characters, but those didn't feel right to me

Example with embedded strings:

```
%b[
  89 "PNG" 0d0a1a0a # PNG header
  0000000d          # Length = 13 bytes
  "IHDR"            # IHDR chunk
  00000060          # Width = 96px
  00000060          # Height = 96px
  08 06             # 8bpp RGBA
  00 00 00          # deflate / no filter / non-interlaced
]
```

Example with interpolation:

```
%b[
  #{png_header}
  #{ihdr = chunk(:ihdr, width: 96, height: 96, bpp: 8, format: :rgba)}
  #{png_crc(ihdr)} # I didn't include this in the other examples but I needed something to demonst
rate here
]
```

Other possible alternatives:

- A library (possibly standard library/gem) could have a function like binary take a string (potentially a heredoc) and parse it according to the same rules I wrote above. You would have to make the parser strip whitespace and comments, and only hex bytes could be interpolated.
- A new pack/unpack symbol could be created that does the same thing as above, so you could ["hex #comments\netc"].pack("...")
- You could probably do a lot of this with an array of hex strings and pack but it doesn't allow for freeform whitespace and the way you do it is not obvious without reading the docs for pack ... and also you allocate a bunch of strings you don't need
- A Data-like object more closely related to IO::Buffer could be defined that declares the size of things contained within a buffer and then a constant could be written to create an instance of the Data-subclass containing the actual data you want to write out ... but this is a lot of work

Potential users:

- People writing protocol code in Ruby
- People who need to write out magic constants (in my case: the RDB encoding of a Redis value)
- People using something like Metasploit Framework to reverse engineer something
- Tools could e.g. disassemble x86 into a literal with comments showing the assembly mnemonics

History

#1 - 01/21/2024 06:21 AM - rubyFeedback (robert heiler)

I think it may be easier to focus only on %b() at first, rather than concomitantly on %B() as well, for returning an IO::Buffer. It could possibly be confusing if there is both %b() and %B() suggested doing different things - that's why I think focusing just on %b() may be easier and reduce the scope (and possible impact); and if %b() finds approval, one can then still focus on %B() via a proposal. But this is just a suggestion - feel free to ignore it if your expectation is different.

If I understand the proposal here correctly, then %b() would allow users to more easily add comments and formatting as to how the user wants binary data to be stored in a .rb file? For sake of completion, could the current way how to handle such data also be shown, for comparison 1:1? That may help folks decide on the objective advantages and trade-offs in regards to the suggested feature.

At the least I find the syntax proposal here nice, so I am +1 on it. It seems useful and possibly an improvement over the status quo, but I rarely deal with binary data in ruby myself, so I may not the best individual to comment on an objective usefulness of the suggested feature at hand.

I guess one additional consideration to make is whether %b() may be used for another feature or functionality in the future (or not), as once when a feature is suggested (for syntax such as %b()), changing it in the future may be a bit problematic due to matz favouring backwards compatibility when possible, so that should be discussed upfront in my opinion.

I'd also suggest to add this to an upcoming developer meeting, after some discussion, to get matz' and the core's team opinion on it (after that discussion).

#2 - 01/21/2024 09:01 AM - retro (Josef Šimánek)

> A library (possibly standard library/gem) could have a function like binary take a string (potentially a heredoc) and parse it according to the same rules I wrote above. You would have to make the parser strip whitespace and comments, and only hex bytes could be interpolated.

Have you checked bindata gem? There is also kaitai project.

#3 - 01/22/2024 06:51 PM - ziggythehamster (Keith Gable)

rubyFeedback (robert heiler) wrote in #note-1:

> I think it may be easier to focus only on %b() at first, rather than concomitantly on %B() as well, for returning an IO::Buffer. It could possibly be confusing if there is both %b() and %B() suggested doing different things - that's why I think focusing just on %b() may be easier and reduce the scope (and possible impact); and if %b() finds approval, one can then still focus on %B() via a proposal. But this is just a suggestion - feel free to ignore it if your expectation is different.

I agree with that. Technically %B[] would be equivalent to IO::Buffer.for(%b[]) and that's not that much extra typing.

rubyFeedback (robert heiler) wrote in #note-1:

> If I understand the proposal here correctly, then %b() would allow users to more easily add comments and formatting as to how the user wants binary data to be stored in a .rb file? For sake of completion, could the current way how to handle such data also be shown, for comparison 1:1? That may help folks decide on the objective advantages and trade-offs in regards to the suggested feature.

You're correct, but even without comment support, having a syntax for this makes it easier to work with small blob constants.

The example that I ran into that I wanted this is that we're writing an application to migrate data from one Redis server to another, and in our case, we only care about the key and the object's idle time. The only way to establish a key in Redis with a correct idle time is to use RESTORE, which requires you to pass something that is serialized in Redis DUMP format. For performance reasons, we don't want to have to dump the constant value of the keys (turns O(N) into O(2N)), so we were going to make a Ruby constant equal to a valid serialization payload. Here's what that looks like:

```
# This doesn't work because the encoding is wrong (defaults to whatever the system encoding is)
REDIS_DUMP_ONE = "\x00\xc0\x01\t\x00\xf6\x8a\xb6z\x85\x87rM"

# This doesn't work because the bare string is treated like it's the system encoding by some tools... also it'
s a bit verbose
REDIS_DUMP_ONE = String.new("\x00\xc0\x01\t\x00\xf6\x8a\xb6z\x85\x87rM", encoding: Encoding::BINARY).freeze

# This works but requires you know what pack does and doesn't allow you to add any annotations
REDIS_DUMP_ONE = ["00c0010900f68ab67a8587724d"].pack("H*").freeze

# If this proposed syntax existed
REDIS_DUMP_ONE = %b[00c0010900f68ab67a8587724d]

# If we wanted to document it using the proposed syntax
REDIS_DUMP_ONE = %b[
  00               # String
  c0               # 8-bit integer
  01               # "1"
  0900             # RDB version
  f68ab67a8587724d # CRC64
]

# Or document with existing syntax
REDIS_DUMP_ONE = [
  [
    "00",              # String
    "c0",              # 8-bit integer
    "01",              # "1"
    "0900",            # RDB version
    "f68ab67a8587724d" # CRC64
  ].join
].pack("H*").freeze
```

Another case where we work with small snippets like this is in specs. We use Kafka with the Confluent Schema Registry wire format, and it's useful to verify that we're trying to write the correct payloads to Kafka. We end up stubbing the schema ID lookups, so we end up with a constant header

"\x00\x00\x00\x00\x01" that is prepended to whatever payload we expect to write. This is super trivial to do in Ruby, but we end up with something like this in our specs so that colleagues are able to figure out what this special constant means:

```
#               magic byte
#                   ||,,,,,,,,------ schema ID (int32)
let(:sr_header) { ["0000000001"].pack("H*").freeze }

# If we wanted to document it using existing syntax
let(:sr_header) do
  [
    [
      "00",       # magic byte
      "00000001" # schema ID
    ].join
  ].pack("H*").freeze
end

# Or if the proposed syntax existed
let(:sr_header) do
  %b[
    00       # magic byte
    00000001 # schema ID
  ]
end
```

rubyFeedback (robert heiler) wrote in #note-1:

> At the least I find the syntax proposal here nice, so I am +1 on it. It seems useful and possibly an improvement over the status quo, but I rarely deal with binary data in ruby myself, so I may not the best individual to comment on an objective usefulness of the suggested feature at hand.
>
> I guess one additional consideration to make is whether %b() may be used for another feature or functionality in the future (or not), as once when a feature is suggested (for syntax such as %b()), changing it in the future may be a bit problematic due to matz favouring backwards compatibility when possible, so that should be discussed upfront in my opinion.
>
> I'd also suggest to add this to an upcoming developer meeting, after some discussion, to get matz' and the core's team opinion on it (after that discussion).

I also infrequently need to do this, but when I do, I find the possible syntaxes to not be as pleasing as they could be. New percent literals are very rarely added, so I don't think there are any backwards compatibility concerns. One way I could see this discussion going is toward a more generic "fold whitespace and support comments" syntax like this (similar to multi-line regexes):

```
%c[
  The quick brown # comment
  fox jumped over # another comment
  the lazy dog.   # a third comment
] # => "The quick brown fox jumped over the lazy dog."
```

This would then allow for this:

```
[%c[
    000102 # foo
    030405 # bar
].delete(' ')].pack("H*").freeze
```

Which is less pretty but could be turned into this with a method like:

```
binstr %c[
  000102 # foo
  030405 # bar
]
```

retro (Josef Šimánek) wrote in #note-2:

> > A library (possibly standard library/gem) could have a function like binary take a string (potentially a heredoc) and parse it according to the same rules I wrote above. You would have to make the parser strip whitespace and comments, and only hex bytes could be interpolated.
>
> Have you checked bindata gem? There is also kaitai project.

Both of these are very useful libraries when you are working with more than a snippet of binary data, but I rarely find myself needing to work with anything more than a short string that is either constant or *mostly* constant. A more formal declaration of data structures and such like the above would definitely be preferred for anything more than snippets of binary data. It's far more frequent that I want to do something like above, or last night I found myself wanting to be able to do this to an x86-16 binary:

```
bin = File.binread('foo.bin')

# convert all jmp +0x00 instructions to two byte nops because Ghidra gets confused:
bin.gsub(%b[eb 00], %b[66 90])

File.binwrite('bar.bin', bin)
```

#4 - 01/24/2024 07:22 AM - shyouhei (Shyouhei Urabe)

Reminds me of discussions in ISO C/C++ committee (in both, simultaneously, by the same person). C23 opted not to have binary string literal, instead introduced a way to load binaries into C source codes by a new preprocessor directive called #embed.

https://en.cppreference.com/w/c/preprocessor/embed

Just for reference.

#5 - 01/27/2024 02:34 AM - ziggythehamster (Keith Gable)

shyouhei (Shyouhei Urabe) wrote in #note-4:

> Reminds me of discussions in ISO C/C++ committee (in both, simultaneously, by the same person). C23 opted not to have binary string literal, instead introduced a way to load binaries into C source codes by a new preprocessor directive called #embed.
>
> https://en.cppreference.com/w/c/preprocessor/embed
>
> Just for reference.

In Ruby's case, any binary data that is large enough that you would want it in a separate file but small enough that you would want it to be in memory can be loaded with File.binread in much the same way. I'm thinking about this more as a usability/beauty improvement over "\xAB\xCD\xEF\x12\x34\x56".force_encoding(Encoding::BINARY).

#6 - 01/29/2024 11:59 PM - shan (Shannon Skipper)

For what it's worth, there's a shorthand String#b.

```
"\xAB\xCD\xEF\x12\x34\x56".b
#=> => "\xAB\xCD\xEF\x124V"
```

For fun, here's a Ruby implementation along the lines of what you propose.

```
module Kernel
  def Binary(string, exception: true)
    hex = string.b
    hex.gsub!(/#[^#]*$\R*/, '')
    hex.gsub!(/"[^"]*"/) do |quotes|
      quotes[1..-2].unpack1('H*')
    end
    hex.delete!("\s\t")

    if hex.match?(/\H/)
      return unless exception

      invalid_chars = hex.scan(/\H/).to_set.join
      raise ArgumentError,
            "invalid non-hex chars for Binary(): #{invalid_chars.inspect}"
    end

    [hex].pack('H*').freeze
  end
end

string = <<-BINARY
  89 "PNG" 0d0a1a0a # PNG header
  0000000d          # Length = 13 bytes
  "IHDR"            # IHDR chunk
  00000060          # Width = 96px
  00000060          # Height = 96px
  08 06             # 8bpp RGBA
  00 00 00          # deflate / no filter / non-interlaced}
```

```
BINARY

binary = Binary(string)
pp string: binary, encoding: binary.encoding, frozen: binary.frozen?
# => {:string=>"\x89PNG\r\n" + "\x1A\n" + "\x00\x00\x00\rIHDR\x00\x00\x00`\x00\x00\x00`\b\x06\x00\x00\x00",
#     :encoding=>#<Encoding:ASCII-8BIT>,
#     :frozen=>true}
```

I like your idea for a nicer binary interface for hex bytes.