Ruby - Feature #20792

String#with_encoding(encoding)

10/09/2024 03:44 PM - kddnewton (Kevin Newton)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

I would like to add a method to String called forcible_encoding?(encoding). This would return true or false depending on whether the receiver can be forced into the given encoding without breaking the string. It would effectively be an alias for:

```
def forcible_encoding?(enc)
  original = encoding
  result = force_encoding(enc).valid_encoding?
  force_encoding(original)
  result
end
```

I would like this method because there are extremely rare but possible circumstances where source files are marked as binary but contain UTF-8-encoded characters. In that case I would like to check if it's possible to cleanly force UTF-8 before actually doing it. The code I'm trying to replace is here: https://github.com/ruby/prism/blob/d6e9b8de36b4d18debfe36e4545116539964ceeb/lib/prism/parse_result.rb#L15-L30.

The pull request for the code is here: https://github.com/ruby/ruby/pull/11851.

History

#1 - 10/09/2024 06:20 PM - Eregon (Benoit Daloze)

I have wanted this feature too, how about adding an optional argument to String#valid_encoding??
Like binary_string.valid_encoding?(Encoding::UTF_8).

#2 - 10/09/2024 06:23 PM - Eregon (Benoit Daloze)

In terms of performance though, both methods need 2 code range scans if the String is then force_encoding(Encoding::UTF_8) if valid and used later for some operation needing the code range.
Maybe the String should stay in the argument encoding if it's valid in it?

#3 - 10/09/2024 06:53 PM - Eregon (Benoit Daloze)

I think I discussed this with @byroot (Jean Boussier) a couple times as well, what about String#with_encoding(Encoding), which is like force_encoding but doesn't mutate the receiver.
Then this would be efficient in that it would scan the code range once:

```
utf8 = binary_str.with_encoding(Encoding::UTF_8)
if utf8.valid_encoding?
  return utf8
else
  return binary_str
end
```

And of course it would shorten str.dup.force_encoding(Encoding::UTF_8) to str.with_encoding(Encoding::UTF_8).

#4 - 10/09/2024 07:07 PM - austin (Austin Ziegler)

Eregon (Benoit Daloze) wrote in #note-3:

> I think I discussed this with @byroot (Jean Boussier) a couple times as well, what about String#with_encoding(Encoding), which is like force_encoding but doesn't mutate the receiver.
> Then this would be efficient in that it would scan the code range once:
>
> ```
> utf8 = binary_str.with_encoding(Encoding::UTF_8)
> if utf8.valid_encoding?
>   return utf8
> else
>   return binary_str
> ```

```
end
```

> And of course it would shorten str.dup.force_encoding(Encoding::UTF_8) to str.with_encoding(Encoding::UTF_8).

A variation of this would be something like:

```
if utf8 = binary_str.try_encoding(Encoding::UTF_8)
  return utf8
else
  return binary_str
end
```

The implementation would be the equivalent of:

```
def try_encoding(encoding)
  target = self.dup.force_encoding(encoding)
  target.valid_encoding? ? target : nil
end
```

#5 - 10/09/2024 10:34 PM - Eregon (Benoit Daloze)

Right, that would also solve this specific usage.
However I think String#with_encoding is a good general method to have, as for instance there are lots of .dup.force_encoding.
It's one of the rare cases where Ruby core API don't have a nice way to do something in a non-mutating manner so that would fix it.
It's especially striking for frozen literal strings where one just wants it in some specific encoding, but currently one has to "foo".dup.force_encoding(SOME_ENCODING).
There is String#b but that's only for the BINARY encoding (and the name is not really clear).

#6 - 10/10/2024 03:44 AM - nirvdrum (Kevin Menard)

Eregon (Benoit Daloze) wrote in #note-1:

> I have wanted this feature too, how about adding an optional argument to String#valid_encoding??
> Like binary_string.valid_encoding?(Encoding::UTF_8).

I'm partial to this one. Alternatively, it could be nice to have the inverse: Encoding#valid_string? (or Encoding#valid_bytes?). I like the symmetry, but you'd give up the short-hand of specifying the encoding by its string name.

#7 - 10/11/2024 08:05 AM - nobu (Nobuyoshi Nakada)

nirvdrum (Kevin Menard) wrote in #note-6:

> I'm partial to this one. Alternatively, it could be nice to have the inverse: Encoding#valid_string? (or Encoding#valid_bytes?).

I prefer valid_sequence? over valid_bytes?.

> I like the symmetry, but you'd give up the short-hand of specifying the encoding by its string name.

Encoding.valid_string?(str, enc) may be possible as a short-hand for Encoding.find(enc).valid_string?(str).

#8 - 10/11/2024 03:04 PM - byroot (Jean Boussier)

> as a short-hand for Encoding.find(enc).valid_string?(str).

I suspect most of the time you'd check a specific encoding, so Encoding::UTF_8.valid_sequence?(str) would be enough.

That said I think that what @Eregon (Benoit Daloze) mentioned in term of performance makes sense. Most of the time when using such method, the next step will be to dup the string and call force_encoding on it.

So a higher level method would have the advantage of being able to store the computed coderange on the resulting string.

#9 - 10/14/2024 05:41 PM - kddnewton (Kevin Newton)

I think the advantage right now is that it doesn't require a mutable string to check. It seems like all of these other options would? Unless you mean to make with_encoding return nil if it wasn't valid?

#10 - 10/14/2024 07:24 PM - Eregon (Benoit Daloze)

> I think the advantage right now is that it doesn't require a mutable string to check.

with_encoding would always be the same as .dup.force_encoding (except slightly more efficient).
It doesn't mutate the receiver.

For the description use case you could then use valid_encoding? like in https://bugs.ruby-lang.org/issues/20792#note-3
That might compute the code range (if not already computed), but that's needed to know if the encoding is valid anyway, and interestingly it will remember this coderange for further operations on that new String.

Using forcible_encoding? it can't remember the code range and so it would have to be recomputed on force_encoding or the next operation needing it.

#11 - 10/17/2024 04:31 PM - kddnewton (Kevin Newton)

Yeah I'm saying it doesn't require a mutable string because it just checks the current string, it doesn't require a dup/allocation. So this avoids having to allocate a new string to check if it's possible.

#12 - 10/21/2024 10:52 AM - Eregon (Benoit Daloze)

Right.
But if it is valid in that encoding, wouldn't you always or almost always then want the String (or a copy of it) in that encoding?
If you do, .with_encoding would be more efficient as it keeps the computed coderange.
If you don't then indeed just a predicate would avoid the String instance allocation (Strings are copy-on-write, so it's just one object allocation, string bytes are not copied).

Do you have an example where you wouldn't want a String in that encoding?
The linked example needs a UTF-8 String on line 19. So with_encoding seems a perfect fit there and more efficient than the predicate (1 vs 2 coderange scans).
The code also has to explicitly workaround force_encoding being inplace (a common inconvenience with force_encoding) on line 27, which with_encoding addresses.

#13 - 11/06/2024 06:21 PM - kddnewton (Kevin Newton)

- Subject changed from String#forcible_encoding? to String#with_encoding(encoding)

Hmm, I see what you mean. In that case I'm convinced. I have changed the title of this request to be String#with_encoding.

#14 - 11/07/2024 10:38 AM - mame (Yusuke Endoh)

Briefly discussed at the dev meeting. If String#with_encoding is the same as .dup.force_encoding, @matz (Yukihiro Matsumoto) said .dup.force_encoding is good enough.

#15 - 11/08/2024 03:45 PM - kddnewton (Kevin Newton)

- Status changed from Open to Rejected

In that case I think I will close this.