# Ruby master - Bug #13220

## Enhance support of Unicode strings manipulation

02/16/2017 10:34 PM - r.smitala (Radovan Smitala)

| | | | |
|---|---|---|---|
| **Status:** | Feedback | | |
| **Priority:** | Normal | | |
| **Assignee:** | | | |
| **Target version:** | | | |
| **ruby -v:** | ruby 2.4.0p0 (2016-12-24 revision 57164) [x86_64-darwin16] | **Backport:** | 2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN |

**Description**

Hi,

last days, Starr Horne posted very interesting testing results about manipulation unicode strings in Ruby 2.4.
And many methods doesn't work as excepted.

Article:

http://blog.honeybadger.io/ruby-s-unicode-support/

### History

#### #1 - 02/17/2017 02:31 AM - shyouhei (Shyouhei Urabe)

*- Status changed from Open to Feedback*

Can you split this request into several ones? Because what this ticket aims is a bit too large and perhaps vague. It is advised that you should create an issue with an obvious goal.

For instance if you believe String#[] understand unicode grapheme clusters (which is what assumption #2 is about), please make a ticket as such.

#### #2 - 02/17/2017 06:33 AM - r.smitala (Radovan Smitala)

Yes i know its little bit large issue.

I'm not sure how to handle it and separate problematic parts into content blocks.
Or just bug by bug what is 33 issues.

It's not my blog post. But when i tried some testing cases they were really wrong and unexpected.

#### #3 - 02/17/2017 07:44 AM - shyouhei (Shyouhei Urabe)

Radovan Smitala wrote:

> It's not my blog post. But when i tried some testing cases they were really wrong and unexpected.

Can you, then, show us your testing cases so that we can look at the "wrong and unexpected" results?

#### #4 - 02/17/2017 08:21 AM - nobu (Nobuyoshi Nakada)

Note that these results are in NFD.
It seems to result as expected by using NFC.

#### #5 - 02/17/2017 11:23 AM - shevegen (Robert A. Heiler)

Radovan Smitala, an example for splitting up into subsections could be seen here:

https://bugs.ruby-lang.org/issues/5481

This would make it easier for the ruby core team to fix any of the issues (if they
are issues at all in the first place that is).

#### #6 - 02/18/2017 02:59 PM - r.smitala (Radovan Smitala)

Shyouhei Urabe wrote:

> Radovan Smitala wrote:
>
> > It's not my blog post. But when i tried some testing cases they were really wrong and unexpected.
>
> Can you, then, show us your testing cases so that we can look at the "wrong and unexpected" results?

This new information appears on blogpost:

> NOTE: After publication, some readers pointed out that many of the failures I mentioned wouldn't have happened if I would have normalized the unicode test strings. This is true. However strings aren't automatically normalized by Ruby or Rails (in any of the apps I tested). These tests were always meant to illustrate the worst-case and I think they're still useful in that regard.

It looks like, author used some non-standard unicode strings.

```
[1] pry(main)> "ä".ord
=> 97
[2] pry(main)> "ä".unicode_normalized?
=> false
[3] pry(main)> "ä".unicode_normalize.ord
=> 228
[4] pry(main)> "ä".ord
=> 228
[5] pry(main)> "ä".unicode_normalized?
=> true
```

Whole issue is just about that Ruby doesn't automatically normalize strings to Unicode.

#### #7 - 02/18/2017 03:27 PM - r.smitala (Radovan Smitala)

I tested all cases with normalized strings and they works except this examples:

"".to_f and other to numeric conversion.

Unicode character is arabic-inding digit one. but i think it is ok because any japan numerals like  (ichi) aren't converted to standard computers numerals.

I think this issue could be closed.

#### #8 - 02/27/2017 10:58 AM - duerst (Martin Dürst)

Nobuyoshi Nakada wrote:

> Note that these results are in NFD.
> It seems to result as expected by using NFC.

This is mostly true, but there are 'visual' characters that cannot be expressed in a single code point in Unicode. As an example:
"q̈".unicode_normalize.gsub("q", "x") # => "ẍ"
(The "q̈" may show with the two dots above the q or after them depending on the font and rendering engine used by your browser or mailer; in my case, the dots appear after, but the cursor moves across the q and the dots with a single key press.)

For many of the tests, applying them to grapheme clusters might work, but there may be languages where it won't be that easy.

Also, I don't understand why the author expects "ä" for "á".next, but is happy for "á".upto("c̈").to_a to cycle through ["ä", "b̈", "c̈"]. Here, the expectations seem to be inconsistent, but it also has to be said that e.g. Swedes would expect "ä".next to be "ö" (see https://en.wikipedia.org/wiki/Swedish_alphabet).

#### #9 - 06/14/2019 08:58 PM - mihao (Michał Kosek)

Most of these test failures are caused by Ruby operating on code points, not grapheme clusters. There are more and more characters that are only expressed by several code points, and they are not limited to obscure cases, such as "q̈". For example, country flags use two code points, which leads to unexpected results:

"🇵🇱".reverse # => 🇱🇵

Normalisation won't help here; we need grapheme clusters:

"🇵🇱".grapheme_clusters.reverse.join # => 🇵🇱

Please consider making string functions and regexes operate on grapheme clusters by default. That's what users want 99% of the time. Code points are hardly ever a useful unit. For example, a user may want to know the number of grapheme clusters or the number of bytes in a string, but it's hard to find a scenario where it's important to know that "🇵🇱" consists of four code points.

By the way, string operations in Swift don't make such surprises: String("□□□□□□□□□".reversed()) # => □□□□□□□□□