Ruby - Feature #19317

Unicode ICU Full case mapping

01/06/2023 03:05 PM - noraj (Alexandre ZANNI)

Status:	Assigned	
Priority:	Normal	
Assignee:	duerst (Martin Dürst)	
Target version:		

Description

As announced in Case Mapping, Ruby support for Unicode case mapping is not complete yet.

Unicode supports in Ruby is pretty awesome, it works by default nearly everywhere, things are implemented the right way and works as expected by the UTRs.

But some features are still missing.

To reach ICU Full Case Mapping support, a few points need to be enhanced.

context-sensitive case mapping

- ❑ cf. Table 3-17 (Context Specification for Casing) of the Unicode standard and ucd/SpecialCasing.txt.

```
"ΣΣ".downcase # returns σσ instead of σς
```

Output examples in ECMAScript:

```
Σ    □□ σ
Σa   □□ σa
aΣ   □□ aς
aΣa  □□ aσa
ΣA   □□ σa
aΣ a □□ aς a
Σ1   □□ σ1
aΣ1  □□ aς1
ΣΣ   □□ σς
```

language-sensitive case mapping

- ❑ Lithuanian rules
- ■ Turkish and Azeri

```
"I".downcase # => "i"
"I".downcase(:turkic) # => "ı"
"I\u0307".upcase # => "I"
"I\u0307".upcase(:lithuanian) # => "I" instead of "I"
```

- ❑ using some standard locale / language codes

Also, it's true that for now there are only a few language-sensitive rules (for Lithuanian, Turkish and Azeri) but why:

- adding a :turkic symbol and not a :azeri?
- using full english arbitrary (why turkic and not turkish?) language name rather than some ICU locale IDs?
  - Language code ISO-639 standard
  - Script code Unicode ISO 15924 Registry
  - country code ISO-3166 standard

So I would rather see something like that

```
"placeholder".upcase(locale: :tr_TR)
"placeholder".upcase(lang: :tr)
```

Related issues:

History

#1 - 01/07/2023 02:03 AM - nobu (Nobuyoshi Nakada)

- Description updated

- Status changed from Open to Assigned

- Assignee set to duerst (Martin Dürst)

#2 - 01/07/2023 11:52 AM - duerst (Martin Dürst)

- Related to Feature #10085: Add non-ASCII case conversion to String#upcase/downcase/swapcase/capitalize added

#3 - 01/07/2023 11:54 AM - duerst (Martin Dürst)

Just answering to one part:

noraj (Alexandre ZANNI) wrote:

> ## language-sensitive case mapping
>
> - ❑ using some standard locale / language codes
>
> Also, it's true that for now there are only a few language-sensitive rules (for Lithuanian, Turkish and Azeri) but why:
>
> - adding a :turkic symbol and not a :azeri?
> - using full english arbitrary (why turkic and not turkish?) language name rather than some ICU locale IDs?
>
> 'turkic' was chosen because it includes both Turkish and Azeri languages (see https://en.wikipedia.org/wiki/Turkic_languages).
>
> - Language code ISO-639 standard
> - Script code Unicode ISO 15924 Registry
>
> Script isn't relevant here, as the characters themselves are directly available.
>
> - country code ISO-3166 standard
>
> So I would rather see something like that
>
> ```
> "placeholder".upcase(locale: :tr_TR)
> "placeholder".upcase(lang: :tr)
> ```

Something like this was discussed. My recollection was that it was rejected because it was overkill for the case at hand, and there was no other functionality in core Ruby that needed it.

#4 - 01/08/2023 12:45 AM - noraj (Alexandre ZANNI)

duerst (Martin Dürst) wrote in #note-3:

> Something like this was discussed. My recollection was that it was rejected because it was overkill for the case at hand, and there was no other functionality in core Ruby that needed it.

Maybe but that would be clearer if other options need to be passed as well, more standard (it could plug well with RDoc::I18n::Locale that already uses IETF BCP 47 language tag or with IRB::Locale rather than having to customly map tr_TR and az_AZ with turkic and lt_LT with lithuanian), this would also map better with locales from the system (eg. /etc/locale.conf, LANGUAGE environment variable).

#5 - 01/08/2023 09:46 AM - zverok (Victor Shepelev)

@noraj I believe the important point here is that there are many **turkic** languages, and as far as I understand, more than two of them use "dotless i".

At least Crimean Tatar (with Latin alphabet) definitely does. Wikipedia lists more active languages using the letter, so in the proposed API, all of them should be accounted for?..

Also, I believe that having a formal language code in the API (instead of a small informal list of writing systems supported) creates a false expectation that every language specificity might be properly accounted for, otherwise "looks like a bug", no?..

```
"STRASSE".downcase(lang: :de_DE)
# => "strasse"
```

```
# But in "properly supported" German, it probably should be
# => "straße"
```

#6 - 01/08/2023 01:24 PM - noraj (Alexandre ZANNI)

zverok (Victor Shepelev) wrote in #note-5:

> Also, I believe that having a formal language code in the API (instead of a small informal list of writing systems supported) creates a false expectation that every language specificity might be properly accounted for, otherwise "looks like a bug", no?..
>
> ```
> "STRASSE".downcase(lang: :de_DE)
> # => "strasse"
> # But in "properly supported" German, it probably should be
> # => "straße"
> ```

No.

1. The correct lower casing for STRASSE is strasse even in german. It's only the other was around that straße should be uppercased to STRASSE. But that is handled by ruby correctly. See Unicode Spec. This is not a reversible operation.
2. ß case mapping is language invariant, it means it should have the same behavior independently of the language.

Is Ruby already using the UCD (Unicode Character Database see UAX #44)?

#7 - 01/08/2023 01:54 PM - zverok (Victor Shepelev)

Oh, OK, I see where you are coming from (the formal correctness of correspondence to Unicode standard/known standard definitions), while I was operating in vague and informal terms "what the user might've meant".

I still don't know what's the "right" way to handle "turkic" problem more formally.
Unicode standards seem to kind of ignore the problem, as far as I can tell, though I am not very well-versed in this. At least SpecialCasing.txt uses the informal term "turkic" in the comments:

```
# Preserve canonical equivalence for I with dot. Turkic is handled below.
```

...but then just introduces two independent lines for Turkish and Azeri, ignoring any other turkic langs:

```
# Turkish and Azeri

# I and i-dotless; I-dot and i are case pairs in Turkish and Azeri
# The following rules handle those cases.

0130; 0069; 0130; 0130; tr; # LATIN CAPITAL LETTER I WITH DOT ABOVE
0130; 0069; 0130; 0130; az; # LATIN CAPITAL LETTER I WITH DOT ABOVE
# ...and so on
```