Ruby master - Bug #4044

Regex matching errors when using \W character class and /i option

11/11/2010 05:08 PM - ben_h (Ben Hoskings)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>naruse (Yui NARUSE)</td>
</tr>
<tr>
<td>Target version:</td>
<td>ruby 1.9.2p0 (2010-08-18 revision 29036) [x86_64-darwin10.4.0]</td>
</tr>
</tbody>
</table>

Description

=begin
Hi all,

Josh Bassett and I just discovered an issue with regex matches on ruby-1.9.2p0. (We reduced it while we were hacking on gemcutter.)

The case-insensitive (/i) option together with the non-word character class (/W/) match inconsistently against the alphabet. Specifically the regex doesn't match properly against the letters 'k' and 's'.

The following expression demonstrates the problem in irb:

puts ('a'..'z').to_a.map { |c| [c, c.ord, c/[^[\W]/i] ].inspect }

As a reference, the following two expressions are working properly:

puts ('a'..'z').to_a.map { |c| [c, c.ord, c/[^[\W]/] ].inspect }
puts ('a'..'z').to_a.map { |c| [c, c.ord, c/[^[\w]/i] ].inspect }

Cheers
Ben Hoskings & Josh Bassett
=end

Related issues:

- Has duplicate Ruby master - Bug #5871: regexp \W matches some word characters... Rejected 01/10/2012
- Has duplicate Ruby master - Bug #7534: /(?i:\[\W\])/ and /(?i:\[\w\])/ both match... Closed 12/08/2012
- Has duplicate Ruby master - Bug #7533: Oniguruma hates the letter 's' :-( Closed 12/08/2012
- Has duplicate Ruby master - Bug #9087: swallowing "s" letters when "i" flag i... Closed 11/06/2013

Associated revisions

Revision d265baf3 - 11/13/2013 12:05 AM - zzak (Zachary Scott)

- doc/regexp.rdoc: [DOC] add note about Bug #4044 as suggested by duerst-san in [ruby-core:43612] [Fixes GH-443] Patched by @rosenfeld
  https://github.com/ruby/ruby/pull/443
  git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@43657 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 43657 - 11/13/2013 12:05 AM - zzak (Zachary Scott)

- doc/regexp.rdoc: [DOC] add note about Bug #4044 as suggested by duerst-san in [ruby-core:43612] [Fixes GH-443] Patched by @rosenfeld
  https://github.com/ruby/ruby/pull/443

Revision 43657 - 11/13/2013 12:05 AM - zzak (Zachary Scott)

- doc/regexp.rdoc: [DOC] add note about Bug #4044 as suggested by duerst-san in [ruby-core:43612] [Fixes GH-443] Patched by @rosenfeld
  https://github.com/ruby/ruby/pull/443

Revision 43657 - 11/13/2013 12:05 AM - zzak (Zachary Scott)

- doc/regexp.rdoc: [DOC] add note about Bug #4044 as suggested by duerst-san in [ruby-core:43612] [Fixes GH-443] Patched by @rosenfeld
  https://github.com/ruby/ruby/pull/443
Bug #4044: Regex matching errors when using \W character class and /i option
http://redmine.ruby-lang.org/issues/show/4044

Author: Ben Hoskings
Status: Open, Priority: Normal
Category: core, Target version: 1.9.2
ruby -v: ruby 1.9.2p0 (2010-08-18 revision 29036) [x86_64-darwin10.4.0]

Hi all,

Josh Bassett and I just discovered an issue with regex matches on ruby-1.9.2p0. (We reduced it while we were hacking on gemcutter.)

The case-insensitive (/i) option together with the non-word character class (/W) match inconsistently against the alphabet. Specifically the regex doesn't match properly against the letters 'k' and 's'.

The following expression demonstrates the problem in irb:

```ruby
puts ('a'..'z').to_a.map { |c| [c, c.ord, c[/[^\W]/i]] }.inspect
```

As a reference, the following two expressions are working properly:

```ruby
puts ('a'..'z').to_a.map { |c| [c, c.ord, c[/[^W]/]] }.inspect
puts ('a'..'z').to_a.map { |c| [c, c.ord, c[/[^\W]/i]] }.inspect
```

Cheers
Ben Hoskings & Josh Bassett

Confirmed on trunk (ruby 1.9.3dev (2010-11-09 trunk 29728) [x86_64-darwin10.4.0]).
And a possible patch is

```c
--- regparse.c 2010-11-12 15:10:07.000000000 +0900
+++ regparse.c.new 2010-11-12 15:29:34.000000000 +0900
@@ -5075,7 +5075,7 @@
     int is_in = onig_is_code_in_cc(env->enc, from, cc);
 #ifdef CASE_FOLD_IS_APPLIED_INSIDE_NEGATIVE_CCLASS
     if ((is_in != 0 && !IS_NCCLASS_NOT(cc)) ||
-        (is_in == 0 && IS_NCCLASS_NOT(cc)))) {
+        (is_in == 0 && IS_NCCLASS_NOT(cc) && from < SINGLE_BYTE_SIZE)) {
         if (ONIGENC_MBC_MINLEN(env->enc) > 1 || *to >= SINGLE_BYTE_SIZE) {
             add_code_range0(&cc->mbuf), env, *to, *to, 0);
         }
```

#4 - 11/14/2010 09:49 PM - naruse (Yui NARUSE)

(2010/11/12 15:36), Heesob Park wrote:

And a possible patch is

```c
--- regparse.c 2010-11-12 15:10:07.000000000 +0900
+++ regparse.c.new 2010-11-12 15:29:34.000000000 +0900
@@ -5075,7 +5075,7 @@
     int is_in = onig_is_code_in_cc(env->enc, from, cc);
 #ifdef CASE_FOLD_IS_APPLIED_INSIDE_NEGATIVE_CCLASS
     if ((is_in != 0 && !IS_NCCLASS_NOT(cc)) ||
-        (is_in == 0 && IS_NCCLASS_NOT(cc)))) {
+        (is_in == 0 && IS_NCCLASS_NOT(cc) && from < SINGLE_BYTE_SIZE)) {
         if (ONIGENC_MBC_MINLEN(env->enc) > 1 || *to >= SINGLE_BYTE_SIZE) {
             add_code_range0(&cc->mbuf), env, *to, *to, 0);
         }
```

Thank you for a patch, but it breaks

"[^u0100]/i =~ "\u0101"

... NARUSE, Yui naruse@airemix.jp

#5 - 11/15/2010 10:05 AM - phasis68 (Heesob Park)

(2010/11/14 NARUSE, Yui naruse@airemix.jp):

And a possible patch is

```c
--- regparse.c 2010-11-12 15:10:07.000000000 +0900
+++ regparse.c.new 2010-11-12 15:29:34.000000000 +0900
@@ -5075,7 +5075,7 @@
     int is_in = onig_is_code_in_cc(env->enc, from, cc);
 #ifdef CASE_FOLD_IS_APPLIED_INSIDE_NEGATIVE_CCLASS
     if ((is_in != 0 && !IS_NCCLASS_NOT(cc)) ||
-        (is_in == 0 && IS_NCCLASS_NOT(cc)))) {
+        (is_in == 0 && IS_NCCLASS_NOT(cc) && from < SINGLE_BYTE_SIZE)) {
         if (ONIGENC_MBC_MINLEN(env->enc) > 1 || *to >= SINGLE_BYTE_SIZE) {
             add_code_range0(&cc->mbuf), env, *to, *to, 0);
         }
```

3/28/2020
Thank you for a patch, but it breaks

/[^\u0100]/i =~ "\u0101"

OK, here is a revised patch

--- regparse.c 2010-11-15 10:02:34.000000000 +0900
+++ regparse.c.new 2010-11-15 10:01:20.000000000 +0900
@@ -5075,7 +5075,9 @@
     #ifdef CASE_FOLD_IS_APPLIED_INSIDE_NEGATIVE_CCLASS
     if ((is_in != 0 && !IS_NCCLASS_NOT(cc)) ||
         (is_in == 0 && IS_NCCLASS_NOT(cc) &&
          ((from < SINGLE_BYTE_SIZE && *to < SINGLE_BYTE_SIZE) ||
            (from >= SINGLE_BYTE_SIZE && *to >= SINGLE_BYTE_SIZE)))) {
         if (ONIGENC_MBC_MINLEN(env->enc) > 1 || *to >= SINGLE_BYTE_SIZE) {
             add_code_range0(&cc->mbuf, env, *to, *to, 0);
         }
     }

Regards,
Park Heesob

#6 - 11/17/2010 07:48 PM - naruse (Yui NARUSE)

It is still a hack.
Current behavior has a reason:
\W -> (ignore case) -> \W (\u017F) + s + S + ... -> not

An experimental patch is following but this is also wrong.

diff --git a/ChangeLog b/ChangeLog
index 18567e3..9dbe329 100644
--- a/ChangeLog
+++ b/ChangeLog
@@ -1,3 +1,8 @@
+**regparse.c:** don't apply ignore case to posix bracket, character
+ type, and character property. [ruby-core:33139]
+ Wed Nov 17 15:16:48 2010 NARUSE, Yui <naruse@ruby-lang.org>
+ Wed Nov 17 17:19:02 2010 NARUSE, Yui <naruse@ruby-lang.org>

* regexp.h (OnigOpInfoType): constify name.
diff --git a/regparse.c b/regparse.c
index bf40601..118081f 100644
--- a/regparse.c
+++ b/regparse.c
@@ -4270,6 +4270,8 @@ code_exist_check(OnigCodePoint c, UChar* from, UChar* end, int ignore_escaped,
          return 0;
 }
+static int cclass_case_fold(Node** np, CClassNode *cc, ScanEnv* env);

+static int
parse_char_class(Node** np, OnigToken* tok, UChar** src, UChar* end,
ScanEnv* env)
@@ -4279,13 +4281,14 @@
 Node* node;
 CClassNode *cc, *next_cc;
 CClassNode work_cc, cased_cc;
- CClassNode work_cc, cased_cc;
+ CClassNode work_cc, cased_cc;

 enum CCSTATE state;
 enum CCLASSVALTYPE val_type, in_type;
 int val_israw, in_israw;
prev_cc = (CClassNode *)NULL;
+ initialize_cclass(&cased_cc);
np = NULL_NODE;
r = fetch_token_in_cc(tok, src, end, env);
if (r == TK_CHAR && tok->u.c == '^' && tok->escaped == 0) {
@@ -406,7 +4409,7 @@ parse_char_class(Node** np, OnigToken* tok, UChar** src, UChar* end, break;
    case TK_POSIX_BRACKET_OPEN:
        r = parse_posix_bracket(cc, &p, end, env);
        r = parse_posix_bracket(&cased_cc, &p, end, env);
@@ -4409,7 +4409,7 @@
            if (r < 0) goto err;
            if (r == 1) { /* is not POSIX bracket */
                CC_ESC_WARN(env, (UChar*)"[");
@@ -4422,7 +4422,7 @@
    case TK_CHAR_TYPE:
        r = add_ctype_to_cc(cc, tok->u.prop.ctype, tok->u.prop.not, env);
        r = add_ctype_to_cc(&cased_cc, tok->u.prop.ctype, tok->u.prop.not, env);
@@ -4436,7 +4436,7 @@
@@ -4504,7 +4504,7 @@ parse_char_class(Node** np, OnigToken* tok, UChar** src, UChar* end, break;
    }
@@ -4549,7 +4549,7 @@
            r = parse_char_class(&anode, tok, &p, end, env);
            if (r != 0) return r;
    next_class:
@@ -4566,13 +4566,13 @@
    }
@@ -5136,6 +5153,32 @@
            if (IS_IGNORECASE(env->option)) {
                cclass_case_fold(np, cc, env);
            }
@@ -5159,6 +5186,32 @@
        if (IS_NOT_NULL(&cased_cc)) {
            r = or_cclass(cc, &cased_cc, env);
            initialize_cclass(&cased_cc);
            if (IS_NOT_NULL(prev_cc)) {
                r = and_cclass(prev_cc, cc, env);
                if (r != 0) goto err;
@@ -5175,6 +5195,13 @@
            i_apply_case_fold(OnigCodePoint from, OnigCodePoint to[]),
        }
    static int
+ cclass_case_fold(Node** np, CClassNode *cc, ScanEnv* env) +{

int r;
IApplyCaseFoldArg iarg;

iarg.env = env;
iarg.cc = cc;
iarg.alt_root = NULL_NODE;
iarg.ptail = &(iarg.alt_root);

r = ONIGENC_APPLY_ALL_CASE_FOLD(env->enc, env->case_fold_flag,
i_apply_case_fold, &iarg);
if (r != 0) {
  onig_node_free(iarg.alt_root);
  return r;
}

if (!IS_NOT_NULL(iarg.alt_root)) {
  Node* work = onig_node_new_alt(*np, iarg.alt_root);
  if (IS_NULL(work)) {
    onig_node_free(iarg.alt_root);
    return ONIGERR_MEMORY;
  }
  *np = work;
}
return r;


static int
parse_exp(Node** np, OnigToken* tok, int term,
          UChar** src, UChar* end, ScanEnv* env)
{

  case TK_CC_OPEN:
  {
    r = parse_char_class(np, tok, src, end, env);
    if (r != 0) return r;
    cc = NCCLASS(*np);
    if (IS_IGNORECASE(env->option)) {
      IApplyCaseFoldArg iarg;

      iarg.env = env;
iarg.cc = cc;
iarg.alt_root = NULL_NODE;
iarg.ptail = &(iarg.alt_root);

      r = ONIGENC_APPLY_ALL_CASE_FOLD(env->enc, env->case_fold_flag,
i_apply_case_fold, &iarg);
      if (r != 0) {
        onig_node_free(iarg.alt_root);
        return r;
      }
      if (IS_NOT_NULL(iarg.alt_root)) {
        Node* work = onig_node_new_alt(*np, iarg.alt_root);
        if (IS_NULL(work)) {
          onig_node_free(iarg.alt_root);
          return ONIGERR_MEMORY;
        }
        *np = work;
      }
    }
  }
  break;
}

def test_caseless_match
  assert_match!/a/iu, "A"
  assert_match!/a-z/iu, "A"
  assert_not_match!/[:lower:]/iu, "A"
assert_not_match(/\p{Ll}/iu, "A")
assert_not_match(/\p{Lower}/iu, "A")
assert_match(/[^\p{Lower}]/iu, "A")
assert_match(/[^\W]/iu, "A")
end

def test_options
assert_equal(Regexp::IGNORECASE, /a/i.options)
assert_equal(Regexp::EXTENDED, /a/x.options)
end

#7 - 12/26/2010 02:57 PM - naruse (Yui NARUSE)
- Status changed from Assigned to Rejected

I think, current behavior is reasonable.

#8 - 02/09/2011 11:32 AM - towfiq (Mark Towfiq)

Yui NARUSE wrote:

I think, current behavior is reasonable.

Perhaps there is a misunderstanding? The current behavior means that \W does not mean [^A-Za-z0-9_] in Ruby 1.9 in some cases. This is a basic functionality - if people cannot trust the Regexp class abbreviations this will be very difficult. This works correctly in Ruby 1.8.7 BTW. I believe this is a critical bug which must be fixed urgently.

Mark Towfiq
CTO, FanSnap

#9 - 02/09/2011 12:35 PM - naruse (Yui NARUSE)

The current behavior means that \W does not mean [^A-Za-z0-9_] in Ruby 1.9 in some cases.

Unicode ignore case breaks it.
http://unicode.org/reports/tr21/

212A; C; 006B; # KELVIN SIGN
00DF; F; 0073 0073; # LATIN SMALL LETTER SHARP S
http://www.unicode.org/Public/UNIDATA/CaseFolding.txt

\W includes U+212A and U+00DF
\i adds U+006B (k) and U+0073 (S) to [\W]
^ reverses the class; it doesn't include k & S.

This works correctly in Ruby 1.8.7 BTW.

1.8 doesn't have Unicode ignore case.

#10 - 01/10/2012 06:10 PM - duerst (Martin Dürst)
- Status changed from Rejected to Open

In reply to my analysis at https://bugs.ruby-lang.org/issues/5871#note-7, Yui Naruse suggested at https://bugs.ruby-lang.org/issues/5871#note-8 that I open this issue rather than #5871, which I'm doing herewith.

Yui also suggested that I propose a concrete plan. My current proposal is that we analyse what casing data is being used in what places when using /i (case insensitive matching) in regular expressions, and that we then fix that. If we don't make progress, I'll also write to the Unicode mailing list to hopefully collect input from other implementers.

By the way, can somebody explain the following difference:

$ ruby -e "puts /[\W]\u1234/i.match('k').inspect"
#<MatchData "k">

$ ruby -e "puts /[\W]\u1234/i.match('k').inspect"
nil

(\u1234 is there just to force the regexp to be in UTF-8.)
I think this is a bug:

```ruby
$ ruby -e "puts /[\W]\u1234/i.match('k').inspect"
#<MatchData "k">
$ ruby -e "puts /[\W]\u1234/.match('k').inspect"
nil
```

Interesting example:

```ruby
% ruby -ve "("a".."z").each { |ch| p /[\W]/i.match(ch) }"
```

We discussed this issue at today's developers' meeting in Akihabara. There was wide consensus among the attendees that it is very strange to have 'k' and 's' included in the set of non-word (\W) characters. Therefore we are sorry, but we don't agree with your https://bugs.ruby-lang.org/issues/4044#note-7.

duerst (Martin Dürst) wrote:

> My current proposal is that we analyse what casing data is being used in what places when using /i (case insensitive matching) in regular expressions, and that we then fix that.

We have discussed this a bit. The first question is what \w should refer to in Ruby. I personally would hope that in the long term, we can move this to include all word characters (i.e. also non-ascii Latin, other scripts, Hiragana, Katakana, Kanji,...). But the general opinion today was that we should keep this as ASCII only currently. Anyway, this bug is independent of this problem, because in both cases, \w includes 'k' and 's', and therefore in both cases, \W must not include 'k' nor 's'.

Also, we noted that regular expression components such as \w or \W should be independent of whether /i is set or not. The reason for that is that \w already takes care of combining lower- and upper-case characters. So there's nothing a /i can improve, and it should not make things worse.

By the way, can somebody explain the following difference:

```ruby
$ ruby -e "puts /[\W]\u1234/i.match('k').inspect"
#<MatchData "k">
$ ruby -e "puts /\W\u1234/i.match('k').inspect"
```
I suspect that this is due to the fact that \W in character classes gets expanded to an actual list of characters (or ranges) before case-extension (/i), whereas \W outside character classes does not get affected by case-extension.

Given the above, I have reopened this bug. I hope to be able to help you over the next two weeks, but I hope you can take the lead.

Regards, Martin.

#15 - 04/28/2012 01:07 AM - Nevir (Ian MacLeod)

One additional note is that this only seems to occur when \W is in a character group:

```ruby
ruby -ve "{"z"..'z'}.each{|ch| p(/\W/i.match(ch)) }"
```

I think I see the misunderstanding: there are multiple characters that render as 'k' and 's'.

K, S, k, s are basic word characters, and so [^\W] should match them (along with all A-Z and a-z):

0x004B (Latin capital letter K)
0x0053 (Latin capital letter S)
0x006B (Latin capital letter k)
0x0073 (Latin capital letter s)

But, I'm not sure how \[^W\] should treat these characters:

0x00DF (Latin small letter sharp s)
0x017F (Latin small letter long s)
0x212A (Kelvin sign)

The important thing is that all the characters in A-Z (0x41-0x5A) & a-z (0x61-0x7A) are word characters, so \[^W\] should match all of them.

Cheers,
Ben

#17 - 12/19/2012 08:25 AM - phluid61 (Matthew Kerwin)
ben_h (Ben Hoskings) wrote:

But, I'm not sure how \[^W\] should treat these characters:

0x00DF (Latin small letter sharp s)
0x017F (Latin small letter long s)
0x212A (Kelvin sign)

Can you just fall back on the Unicode categories? If we define "word characters" as Letters and Numbers, U+212A is {Lu} and thus a word character. Similarly U+017F is {Ll}.

Seems a bit weird in the case of Kelvin (also the Angstrom Sign U+212B = {Lu}) but at least Unicode is a fixed and universally accessible standard.

#18 - 02/26/2013 10:17 AM - naruse (Yui NARUSE)
- Target version changed from 1.9.2 to 2.6

#19 - 11/07/2013 09:50 PM - rosenfeld (Rodrigo Rosenfeld Rosas)
Shouldn't this bug be mentioned in the docs for \W in the Regexp documentation?
http://www.ruby-doc.org/core-2.0.0/Regexp.html

People would like to be aware of it until it's fixed.

#20 - 11/12/2013 04:53 PM - duerst (Martin Dürst)
On 2013/11/07 21:50, rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

Issue #4044 has been updated by rosenfeld (Rodrigo Rosenfeld Rosas).

Shouldn't this bug be mentioned in the docs for \W in the Regexp documentation?

http://www.ruby-doc.org/core-2.0.0/Regexp.html

People would like to be aware of it until it's fixed.

I'd really prefer it to be fixed, but if you want to contribute a patch on the docu, that would help.

Regards, Martin.

#21 - 11/12/2013 09:34 PM - rosenfeld (Rodrigo Rosenfeld Rosas)
https://github.com/ruby/ruby/pull/443

#22 - 11/13/2013 09:05 AM - zzak (Zachary Scott)
- Status changed from Feedback to Closed
- % Done changed from 0 to 100

This issue was solved with changeset r43657.
Ben, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.
• doc/regexp.rdoc: [DOC] add note about Bug #4044 as suggested by duerst-san in [ruby-core:43612] [Fixes GH-443] Patched by rosenfeld (Rodrigo Rosenfeld Rosas)https://github.com/ruby/ruby/pull/443

#23 - 11/13/2013 09:06 AM - zzak (Zachary Scott)
- Status changed from Closed to Feedback
- % Done changed from 100 to 0

Ooops, didn't mean to close this only mention..

#24 - 08/09/2014 12:37 AM - k_takata (Ken Takata)
I have updated ruby-2.x branch in my Onigmo repository.
I think this bug is fixed now.

(?i)[p][ASCII], (?i)[[:ascii:]], (?ia)[w]. other POSIX classes with (?ia) flags and their negated patterns should not be case folded across ASCII/non-ASCII boundary.
So I make another char class which doesn't include those special patterns. When case folding the original char class, each character is checked whether it is contained in the special char class.

See also https://github.com/k-takata/Onigmo/issues/4, Test patterns are listed. (And more detail is written in Japanese ;-) )

#25 - 09/23/2014 04:51 AM - k_takata (Ken Takata)
- Status changed from Feedback to Closed

Fixed with r47598.

#26 - 01/06/2016 11:26 PM - same (Sam Eaton)
I am experiencing this issue with Ruby 2.3.0 on both OS X 10.10.5 and Ubuntu 14.04.3. When i have a double "f" i get a regex match with the non-word symbol and case insensitivity.

/\W/ =~ "00FF00"     # nil
/\W/i =~ "00FF00"    # 2

#27 - 01/07/2016 04:50 AM - naruse (Yui NARUSE)
Sam Eaton wrote:

/\W/ =~ "00FF00"     # nil
/\W/i =~ "00FF00"    # 2

It's spec.
Its mechanism is, \W includes U+FB00 (LATIN SMALL LIGATURE FF).
\i option expands it into FF.
The the "FF" match given string.

#28 - 01/07/2016 07:59 AM - same (Sam Eaton)
Hmmm... When I try it with any other combination it never matches. Its only when I add the /i then it doesn't matter which case of "f"

"ffffFFFF".scan(/\W/)  # []
"ffffFFFF".scan(/\W/i)  # ["FF", "FFFF", "FF", "FF"]
"ffffFFFF".scan(/\W/i)  # ["FF", "FF", "FF", "FF"]
"ffffFFFF".scan(/\W+/i)  # ["FFFFFFFF"]

I tested these regular expressions with other languages (PHP, Python, JavaScript) and the result was as I expected, no matches. However when I test with Ruby the regex matches. Bug or not, I would hope this could be changed. :) 

#29 - 02/03/2016 03:21 AM - phluid61 (Matthew Kerwin)
I want to write a spec for this, but some of the details are unclear to me. Can we confirm whether each of the following are spec?

RUBY_VERSION #=> "2.3.0"

# eszett (case conversion => multiple chars)
/\W/  =~ "\u00DF" #=> 0
/\W/i =~ 'SS' #=> nil
/\W/i =~ 'SS' #=> 0
I want to write a spec for this, but some of the details are unclear to me. Can we confirm whether each of the following are spec?

Please don’t just assume that the current behavior is spec. If it doesn’t match with common sense in any way, it’s very clear that we have to fix it. There may be borderline cases that are up for discussion, but at least most of the examples I have seen don’t meet that criterion.

My understanding was that Ken Takata fixed the problem with r47598, but I’ll try to have another look at that.

When I looked at Ken’s solution last time (the details are at the following link, in Japanese https://github.com/k-takata/Onigmo/issues/4), it included some aspects related to ASCII, which keeps confusing me.

The relevant specification is Unicode Technical Standard #18, Unicode Regular Expressions, in particular http://www.unicode.org/reports/tr18/#Simple_Loose_Matches. There are various choices at the end of that section that are relevant to this issue.

My personal preference among the choices A-D is B. As far as I understand it, it would mean that while a /i option would change how literal characters are matched, it would not affect how it affects properties such as \W.

My justification for this is as follows: If I want e.g. a word character, then that already should include all the necessary characters, both upper and lower case (and title case just in case you forgot about it :-)). It’s difficult to see why I’d want the set of characters to change when adding /i. The same argument can be applied to \W and most if not all similar cases.

The case that I think can be up for discussion is explicit character classes, such as [a-z]. Here, in effect automatically adding A-Z (and some other case equivalents) may indeed make sense.

Martin Dürst wrote:

On 2016/02/03 12:21, matthew@kerwin.net.au wrote:

    I want to write a spec for this, but some of the details are unclear to me. Can we confirm whether each of the following are spec?

    Please don’t just assume that the current behavior is spec.

Indeed, that’s why I asked.
If it doesn't match with common sense in any way, it's very clear that we have to fix it. There may be borderline cases that are up for discussion, but at least most of the examples I have seen don't meet that criterion.

Confusion abounds. I thought that if there was a formal spec, at least that would give a solid grounding to start from. As it is we rely on implementations to describe what should/does happen, which is imperfect and allows us to confuse bugs with spec.

(Right now I'm particularly interested in why \([\W]i =~ 'k' \#=- nil\))

My understanding was that Ken Takata fixed the problem with r47598, but I'll try to have another look at that.

When I looked at Ken's solution last time (the details are at the following link, in Japanese https://github.com/k-takata/Onigmo/issues/4, it included some aspects related to ASCII, which keeps confusing me.

I've looked at that issue, but I'm afraid I can't read Japanese (and Google translate only gets me so far.) I think I get the gist of it, but any subtlety is probably lost to me.

The relevant specification is Unicode Technical Standard #18, Unicode Regular Expressions, in particular http://www.unicode.org/reports/tr18/#Simple_Loose_Matches. There are various choices at the end of that section that are relevant to this issue.

My personal preference among the choices A-D is B. As far as I understand it, it would mean that while a /i option would change how literal characters are matched, it would not affect how it affects properties such as \W.

I suppose we're in choice D at the moment (that would explain why \W/i and \[\W]i match differently,) but just which "specific properties and/or explicit character classes" remains unclear. Documenting those (and writing a spec) would help.

My justification for this is as follows: If I want e.g. a word character, then that already should include all the necessary characters, both upper and lower case (and title case just in case you forgot about it :-). It's difficult to see why I'd want the set of characters to change when adding /i. The same argument can be applied to \W and most if not all similar cases.

When we were discussing it on Ruby Talk the other day I came up with this:

- the 'ff' ligature is a non-word character
  - it has a case conversion, so is affected by the /i flag

So:

- /ff/ is a subset of \W
  - /ff/ matches 'ff', 'FF', 'ff', 'fF', and 'Ff'
  - therefore \W/i should match all of the above

The first two dot points are where I see the contention. If I were to make a general rule, I'd say that "\W" should not be expanded for case-folding, since 'case' is a property of word characters. (If anything matches "\W" it is, by definition, not a word character, so should not be subject to word-type operations like case-folding.)

If that were so, despite /ff/ =~ 'FF', /\W/i would match 'ff' but not 'FF'.

That would, I think, make \W a perfect complement to \w (identical to \[\w\]); which seems to be what people expect.

I think that means you and I are saying the same thing, in different ways.

The case that I think can be up for discussion is explicit character classes, such as [a-z]. Here, in effect automatically adding A-Z (and some other case equivalents) may indeed make sense.

Certainly; I use /[0-9a-f]/i myself for matching hexadecimal numbers (and similar patterns for similar things.) However where would that leave us with /[a-e]/W i ?