Ruby master - Bug #7342
String#<=> checks for a #to_str method on other but never uses it?
11/13/2012 11:24 AM - jballanc (Joshua Ballanco)

<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>nobu (Nobuyoshi Nakada)</td>
</tr>
<tr>
<td>Target version:</td>
<td>2.0.0</td>
</tr>
<tr>
<td>ruby -v:</td>
<td>2.0.0</td>
</tr>
</tbody>
</table>

**Description**

=begin
This isn't exactly a bug, as much as a request for clarification. I was looking at the semantics of the (({<=>})) operator and noticed something curious. For most classes, when evaluating (({thing <=> other})), if (({other})) is not of a compatible type, then (({nil})) is returned.

The exceptions (as far as I can find) are String and Time. For the Time class, if (({other})) is not a kind of (({Time})), then the reverse comparison (({other <=> thing})) is tried and the inverse of this result is returned (if not nil). For String, the reverse comparison is only tried IF (({other.respond_to?(:to_str)})), HOWEVER the referenced (({other.to_str})) method is never called. For example:

class NotAString
def <=>(other)
  1
end
def to_str
  raise "I'm not a string!"
end
end

"test" <=> NotAString.new #=> -1

This seems very counterintuitive to me. I would expect that if my class implemented (({to_str})), that the return value of this would be used for comparison.
=end

**Associated revisions**

Revision 020cc0ad - 11/30/2012 08:43 AM - nobu (Nobuyoshi Nakada)
string.c: compare with to_str

- string.c (rb_str_cmp_m): try to compare with to_str result if possible before calling <=> method. [ruby-core:49279] [Bug #7342]

Revision 38044 - 11/30/2012 08:43 AM - nobu (Nobuyoshi Nakada)
string.c: compare with to_str

- string.c (rb_str_cmp_m): try to compare with to_str result if possible before calling <=> method. [ruby-core:49279] [Bug #7342]

Revision 38044 - 11/30/2012 08:43 AM - nobu (Nobuyoshi Nakada)
string.c: compare with to_str

- string.c (rb_str_cmp_m): try to compare with to_str result if possible before calling <=> method. [ruby-core:49279] [Bug #7342]

Revision 38044 - 11/30/2012 08:43 AM - nobu (Nobuyoshi Nakada)
string.c: compare with to_str

- string.c (rb_str_cmp_m): try to compare with to_str result if possible before calling <=> method. [ruby-core:49279] [Bug #7342]

Revision 38044 - 11/30/2012 08:43 AM - nobu (Nobuyoshi Nakada)
string.c: compare with to_str
• string.c (rb_str_cmp_m): try to compare with to_str result if possible before calling <=> method. [ruby-core:49279] [Bug #7342]

Revision 38044 - 11/30/2012 08:43 AM - nobu (Nobuyoshi Nakada)
string.c: compare with to_str

• string.c (rb_str_cmp_m): try to compare with to_str result if possible before calling <=> method. [ruby-core:49279] [Bug #7342]

Revision 38044 - 11/30/2012 08:43 AM - nobu (Nobuyoshi Nakada)
string.c: compare with to_str

• string.c (rb_str_cmp_m): try to compare with to_str result if possible before calling <=> method. [ruby-core:49279] [Bug #7342]

History

#1 - 11/14/2012 11:53 AM - jballanc (Joshua Ballanco)
I would expect something like the following patch makes more sense?

diff --git a/string.c b/string.c
index c63f59a..c9eed27 100644
--- a/string.c
+++ b/string.c
@@ -2385,8 +2385,12 @@ rb_str_cmp_m(VALUE str1, VALUE str2)
 long result;
 if (!RB_TYPE_P(str2, T_STRING)) {
     if (!rb_respond_to(str2, rb_intern("to_str"))) {
-        return Qnil;
+        return Qnil;
         if (rb_respond_to(str2, rb_intern("to_str"))) {
             VALUE tmp = rb_funcall(str2, rb_intern("to_str"), 0);
             if (!RB_TYPE_P(tmp, T_STRING)) {
-                return Qnil;
+                return Qnil;
-            } else if (!rb_respond_to(str2, rb_intern("<=>"))) {
+                } else if (!rb_respond_to(str2, rb_intern("<=>"))) {
                 return Qnil;
             }
             result = rb_str_cmp(str1, tmp);  }
         } else if (!rb_respond_to(str2, rb_intern("<=>"))) {
             return Qnil;

#2 - 11/12/2012 12:40 PM - bitsweat (Jeremy Daer)
When an object responds to #to_str, it's saying "I am a string." When an object responds to #to_s, it's saying "I have a string representation."

So checking for #to_str here is enough to know whether str2 is a string and can be compared.

For more background on implicit vs explicit coercion in Ruby: http://briancarper.net/blog/98/

#3 - 11/14/2012 05:55 PM - jballanc (Joshua Ballanco)
As the page you linked points out, #to_str is an implicit cast. i.e. It should be used internally to retrieve the string representation of an object. I think this is in keeping with all other uses of #to_str in Ruby source.

Another thing to note is that currently in Ruby if you have an object that provides #to_str but not #<=>, then it cannot be compared to a string object.

class Foo
def to_str
"my string"
end
end

test" < Foo.new #> ArgumentError: comparison of String with Foo failed

#4 - 11/15/2012 07:57 AM - bitsweat (Jeremy Daer)
"It should be used internally to retrieve the string representation of an object." That's explicit coercion. Implicit coercion with #to_str means the object acts a string and the method needn't be called.

This method is used for more than its return value. It's in a strange limbo world between Ruby and the C API :)

The presence of #to_str indicates that the object obeys an entire String contract such that the C API can work with the object without making Ruby method calls. You note correctly that providing #to_str but not #<=> prohibits comparison. That's because by omitting #<=> you've already broken the "I am a string" contract.

Check out how time.c for another example of checking #to_str and, more generally, see rb_check_convert_type for many other examples of implicit coercion in practice: to_path, to_int, to_ary, etc.
The presence of #to_str indicates that the object obeys an entire String contract such that the C API can work with the object without making Ruby method calls.

Hmm... I was always under the impression that the distinction between #to_s and #to_str is that #to_s provides a (potentially lossy) string representation of any object, but #to_str will return a "string equivalent" of the object. As for the C API, the rb_str_to_str method does call #to_str if v#to_str exists and v is not already a string. I guess it would be good to get some clarification on this issue.

You can use rb_check_funcall().

Thank you for the pointer, nobu! Actually, in looking at the implementation of String#<=> again I found some other oddities. For example, if Other#to_str is defined and Other#<=> returns a float, then "a string" <=> Other.new will return a float. I feel like this breaks the contract of #<=> as it should only ever return 1, 0, or -1. Anyhow, I’ve attached an updated patch that also includes some test fixes.

(Note: all tests in make test-all that passed before this patch pass after, however rubyspec will need to be updated. I will send a pull-request directly to the rubyspec project if this gets accepted.)

This issue was solved with changeset r38044.

Joshua, thank you for reporting this issue.

Your contribution to Ruby is greatly appreciated.

May Ruby be with you.