Ruby trunk - Feature #7849

Symbol#to_str

02/14/2013 04:46 AM - trans (Thomas Sawyer)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:	2.6	

Description

Even though a Symbol is not technically an honest-to-goodness String, from the standpoint of simple practicality it would help to have Symbol#to_str defined.

There are times when we want an argument to accept a String or a Symbol, but don't really want it to accept any type of object under the sun that responds to #to_s --which is just about anything. This is especially the case when writing DSLs. Having Symbol#to_str is the nice solution to this.

Defining Symbol#to_str may be an exception to the rule, but it's one worth making.

History

#1 - 02/14/2013 09:49 AM - Student (Nathan Zook)

Bad idea. to_str should only be defined on things that really are Strings, and Symbol are most definitely not Strings.

I agree that Symbol is unusually close to String. If, for your needs, you were to define to_st on String & on Symbol, you could have the utility you desire.

#2 - 02/14/2013 10:14 AM - trans (Thomas Sawyer)

> If, for your needs, you were to define to_st on String & on Symbol, you could have the utility you desire.

Yes, I thought about that. But concluded it was most likely unnecessary complexity when #to_str would work fine.

You say "Bad idea". But show me why it is bad idea other then "them's the rules". I tried to think of a problem case, and the only one I can think of is using foo.respond_to?(:to_str) to identify Stringy things and very specifically not meaning to include Symbols. It's possible, but it's a fairly narrow proposition. Not the least reason being that one should never use respond_to? if one does not need to b/c it is a fragile approach. But more significantly, what is more likely to be used? This narrow usecase or Symbol#to_str? Clearly the later by far. And the former is easily solved with && !Symbol === foo.

#3 - 02/14/2013 10:36 AM - charliesome (Charlie Somerville)

Symbols are not Strings. I'm afraid this would only serve to blur the line even more.

Rubyists need to stop using Symbols where they actually want a String, and vice versa.

Strong -1 from me.

#4 - 02/14/2013 01:08 PM - drbrain (Eric Hodel)

- Status changed from Open to Rejected

You cannot gsub, enumerate characters in or alter encoding of a Symbol, so it is not a string representation.

#5 - 02/15/2013 09:24 PM - trans (Thomas Sawyer)

> You cannot gsub, enumerate characters in or alter encoding of a Symbol, so it is not a string representation.

That the official spec on the definition of a Stringy-thing? That's the "problem" with #to_str, #to_ary, etc. isn't it? There *is no* absolute interface that dictates their proper use. As long the method returns the expected type then its purely a question of *practicality*. And I submit that Symobol#to_str is about a practical as it gets.

And let me put it another way. If you inherited some code that relied on an object responding to #to_str to ensure it also responded to #gsub, #map

and #force_encoding (which is the crux of your "definition"), what would you think? Yes, you'd have seriously fragile code on your hands and you'd be a'fixing it.

I think you rejected this issue far too prematurely. Do you guys even know the purpose of dialog?

#6 - 02/15/2013 09:58 PM - trans (Thomas Sawyer)

=begin
charliesome (Charlie Somerville) Actually, that's exactly what my proposal attempts to address. You don't always have a choice in what type of object you receive, but you want it to become a string. Consider a DSL like Rake's. One could use:

```
task :foo do ...
```

Or

```
task 'foo' do ...
```

Either one is acceptable, and I think it would be overreaching to make people not be able to use a symbol here.

On the other hand do we want any object to be acceptable? B/c just about every object responds to #to_s. To avoid this, you would end up with something like: (WARNING! Fugly code ahead.)

def task(name)
name = (Symbol === name ? name.to_s : name.to_str)
...
end

There has to be a clearer solution than that.

P.S. Just for fun of it I tried this on rake and discovered the Jim decided not to care what get's passed to task. Try this in your Rakefile:

```
desc "OMG!"
task Object.new do
  puts "OMG is right!"
end
```

A Duck-typing true beleiver!!! Yea, looks like a bug to me. If the user really needs it they can call #to_s.
=end

#7 - 02/16/2013 04:55 AM - drbrain (Eric Hodel)

The purpose of to_str, to_int, to_ary and to_sym are to convert string, integer, array and symbol representations to objects of that class.

For example:

The rope data structure (which supports insertion, deletion and random access) can be used to implement a representation of a ruby string so it would be a good candidate for to_str.

A linked-list implementation could be a good candidate for to_ary

A roman numeral implementation that does not descend from Numeric represents an integer and would be a good candidate for to_int

A string can be used as an identifier (as in rake) so it has to_sym.

A symbol, being an identifier alone is not anything like a String.

#8 - 03/07/2013 10:05 AM - trans (Thomas Sawyer)

Symbol's not anything like a Proc either, but we have Symbol#to_proc.

Put that in your pipe and smoke it ;-)