Postfix `!?` can use as the valid function identifier.

05/15/2015 02:31 PM - make_now_just (Kitsune TSUYUSATO)

Status: Feedback
Priority: Normal
Assignee: 
Target version: 

Description
This patch make it possible to parse and execute such a code:

def foo!? 
  true
end
foo!?

When is this patch useful?
Let's say, there are two methods. One method named ExampleDB#exists?(id) which is to request SQL has I/O side effects, another method named ExampleCollection#exists?(id) which is to find id in memory has no I/O side effects. Calling costs of these methods are different, but we can't know it by code on calling #exists.
In general case, when we want to make the method having to modify object and return boolean value, this patch is useful. Of course such a method is not good, however we are unavoidable to write good efficient program, aren't we?

What effects dose this change have?
For example, snippet foo!?1:2 is interpreted to call method foo! and ternary operation now, but after applying this patch it raises syntax error. Although we can escape this problem by spacing between foo! and ?.
I think that this problem is tiny, but it is incompatible change.

Why is postfix !? only? I want to use ?!, !!? or !?!?
This is the religious problem. If you are Python user, you believe Zen of Python:

  There should be one-- and preferably only one --obvious way to do it.

Or if you are Perl user, you believe TMTOWTDI:

  There's more than one way to do it.

I am not Perl nor Python user. But, this issue's purpose is naming identifier more clearly, so I adopted only !?.

History
#1 - 05/18/2015 04:49 PM - matz (Yukihiro Matsumoto)
- Status changed from Open to Feedback

Hi,

First of all, this is a proposal for incompatible change, so we have to be very careful.
Besides that, I am afraid you have two misunderstandings. (a) in Ruby, "!" is not for methods with side effect, unlike Scheme. (b) predicates (methods with "?") should not have side effect, this is a rule of thumbs.

In summary, we have to reject the proposal, unless you have unseen use-cases.

Matz.