Ruby - Feature #7872

`block_given?` does not work inside `define_method`

02/17/2013 10:40 AM - alexeymuranov (Alexey Muranov)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	2.6	

Description

=begin
Is this the expected behavior?

```
define_method :try do
block_given? ? yield : 'no block'
end

try { 'block' } # => "no block"
```

However:

```
def try_again
block_given? ? yield : 'no block'
end

try_again { 'block' } # => "block"
```

=end

History

#1 - 02/17/2013 01:16 PM - drbrain (Eric Hodel)

- Tracker changed from Bug to Feature

- Target version set to 2.6

=begin
The behavior in 1.9:

```
$ ruby19 -ve 'class C; define_method :x do p block_given? end; end; C.new.x { }'
ruby 1.9.3p374 (2013-01-15 revision 38858) [x86_64-darwin12.2.1]
false
```

Is the same as in 2.0:

```
$ ruby20 -ve 'class C; define_method :x do p block_given? end; end; C.new.x { }'
ruby 2.0.0dev (2013-02-08 trunk 39138) [x86_64-darwin12.2.1]
false
```

So I have switched it to a feature request.
=end

#2 - 02/17/2013 09:26 PM - alexeymuranov (Alexey Muranov)

Ok. Is it actually possible to somehow force def ... end for instance methods behave identically with define_method method with a block?

#3 - 02/18/2013 09:15 AM - ko1 (Koichi Sasada)

- Assignee set to matz (Yukihiro Matsumoto)

(a) def...end and (b) define_method(...){...} is completely different.

(1) On (b), outer scope

```
a = 1
define_method(:foo) do
p a # access to outer scope
```

end

(2) (1) means that the passed block is outer block

```
class C; end
def def_method mid
C.module_eval{
define_method(mid) do
p block_given?
yield if block_given?
end
}
end

def_method(:foo)
obj = C.new
obj.foo
obj.foo{p 1}

def_method(:bar){p :def_foo}
obj.bar
obj.bar{p 2}

#=>

false
false
true
:def_foo
true
:def_foo
```

(3) You can pass block using block parameter

```
define_method(:foo){|&b|
p [b, block_given?]
}
foo   #=> [nil, false]
foo{} #=> [#Proc:0x22d08f0@t.rb:5, false]
```

#4 - 02/18/2013 06:43 PM - alexeymuranov (Alexey Muranov)

@ko1 (Koichi Sasada) thanks for the explanations, i will think about them.

#5 - 04/13/2013 12:43 AM - rkh (Konstantin Haase)

Rebinding block_given? on define_method might be confusing, as the block might be passed to an API without the user being aware of it being used with define_method.

#6 - 04/13/2013 04:14 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Rejected

I'll mark this request as rejected, as it appears based on the misconception that block_given? was false while yield would actually succeed; both refer correctly to the outerscope's presence of the block and arguments, including the block, must be declared explicitly as Koichi points out.

Moreover the request is woefully incomplete as it stands.
If someone feels like there is a feature to be requested, a sensible and more complete proposal must be made, in particular saying if all of block_given?, yield, Proc.new, eval(...), etc..., should refer to the inner scope and why, how this would affect define_method(:foo, &block) (where block is defined somewhere else; would block_given? & al. be magically rebound?), would it apply to define_singleton_method, etc..., why that would be a good thing and what kind of incompatibilities we should expect.