### Ruby master - Feature #3768

**Constant Lookup doesn't work in a subclass of BasicObject**

08/31/2010 04:08 AM - trans (Thomas Sawyer)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>matz (Yukihiro Matsumoto)</td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

```
ruby-1.9.2-p0 > module M
ruby-1.9.2-p0 ?>  end
=> nil
ruby-1.9.2-p0 > class X < BasicObject
ruby-1.9.2-p0 ?>   include M
ruby-1.9.2-p0 ?>   end
NameError: uninitialized constant X::M
from (irb):4:in <class:X>'
from (irb):3
from /home/trans/.rvm/rubies/ruby-1.9.2-p0/bin/irb:17:in'
=end
```

**Related issues:**

Related to Ruby master - Bug #5067: BasicObject's constant lookup documentation

| Closed | 07/21/2011 |

**History**

**#1 - 08/31/2010 11:51 AM - usa (Usaku NAKAMURA)**

- Status changed from Open to Assigned
- Assignee set to matz (Yukihiro Matsumoto)

```
=begin
I think it's spec, but we should hear the opinion of matz.
=.end
```

**#2 - 08/31/2010 02:05 PM - matz (Yukihiro Matsumoto)**

- Status changed from Assigned to Rejected

```
=begin

=.end
```

**#3 - 08/31/2010 02:25 PM - matz (Yukihiro Matsumoto)**

```
=begin

Hi,

BasicObject does not inherit from Object, where the constant M is defined. So, if you want to refer the toplevel constant M, try ::M.

matz.
```

In message "Re: [ruby-core:31956] [Ruby 1.9-Bug#3768][Open] Constant Lookup doesn't work in a subclass of BasicObject" on Tue, 31 Aug 2010 04:08:13 +0900, Thomas Sawyer redmine@ruby-lang.org writes:

```
| Bug #3768: Constant Lookup doesn't work in a subclass of BasicObject
| http://redmine.ruby-lang.org/issues/show/3768
```

```
ruby-1.9.2-p0 > module M
ruby-1.9.2-p0 ?>   end
| => nil
|ruby-1.9.2-p0 > class X < BasicObject
|ruby-1.9.2-p0 ?>   include M
|ruby-1.9.2-p0 ?>   end
```
I see the technical reason it occurs, but to accept that as proper behavior is going to hobble the usefulness of BasicObject.

First of all, it means one's ability to open a class and modify it will be conditional. One will have to check if it is a BasicObject upfront. That's easy to do if you're working with one class you already know, but consider how it effects doing some meta-programming where code is injected into any arbitrary class.

Worst still is that it makes importing code into a namespace very fragile. Consider the simplistic example of having some code in a script to eval into a module.

```ruby
module M
  eval(File.read('file.rb'))
end
```

If `file.rb` contains:

```ruby
class R
end
class Q < BasicObject
def r; R.new; end
end
```

Then it will break whether we use `R` or `::R`.

I feel the underlying issue here goes back to some other issues we've discussed some years ago about the top-level. Routing the toplevel to Object is not as flexible or robust as having a toplevel be an independent self-extended module in which constant resolution would terminate.

How can this be rejected? The example I gave is a glaring problem.

Haven't I explained the reason?

The M is defined under the Object class. The BasicObject does not inherit from Object. So there's no reason M can be accessed from BasicObject, under the current behavior of constant accessing in Ruby. If you want to "fix" this problem, how should I? Making constants under Object can be accessed from everywhere? Or otherwise?

In any case, the "fix" would be huge change to the constant access system, and would introduce huge risk of incompatibility.

I am not sure that a fix is such a huge change. Look-up can be delegated:

```ruby
class BasicObject
  def self.const_missing(name)
    ::Object.const_get(name)
  end
end
```

But yes, I think the ultimate fix does need a rework for constant lookup to terminate at toplevel instead of Object, but I can understand that's a "Ruby 2.0" kind of change. But the above may suffice in the mean time, if it doesn't present any unintended consequences (I can't think of any myself).
While safer, I do not advocate such an approach. For one, there's a TOCTOU race condition in threaded code if Object.remove_const is used. Personally, I don't see this as a major issue. There should be no need for this in BasicObject itself, and overriding const_missing in a BasicObject subclass is easy.

#9 - 07/04/2011 10:48 AM - trans (Thomas Sawyer)
@Jeremy The very need of it is why I reported the issue.

The behavior is clearly broken. It's a pretty fundamental expectation that a subclass of BasicObject would have working constant lookup. To think otherwise is to assert that no subclass of BasicObject should ever be allowed to use delegation.

All I can say is thank goodness for const_missing, b/c if it wasn't for that BasicObject would be all but useless and I literally would not have been able to make two of my programs work correctly (well, without defining my own sub-optimal "BlankSlate" class).

At the VERY least add the work-around to BasicObject's documentation so others will know what to do when their code doesn't work.

#10 - 07/04/2011 11:16 AM - jeremyevans0 (Jeremy Evans)
I disagree that the behavior is "clearly broken". Just like methods defined in Object don't apply to BasicObject, you shouldn't expect constants defined in Object to apply to BasicObject.

You assume that normal constant lookup is always desired in BasicObject subclasses. While true in some cases, it is not necessarily true in all. Take this simple case:

```ruby
class S < BasicObject
def method_missing(m) m end
def self.const_missing(m) m end
end
```

Basically, the programmer desires both that both method calls and constant references return symbols:

```ruby
S.new.instance_eval{puts} # => :puts
S.new.instance_eval{Object} # => :Object
```

With your approach, you would get ::Object instead of :Object for the second line. Just like the puts method doesn't exist in BasicObject instances, the Object constant doesn't exist in BasicObject.

Your recommendation would remove the ability programmers currently have to choose how to implement constant lookup in their BasicObject subclasses. Your recommendation assumes that all users want normal constant lookup in a BasicObject subclass. However, the fact that they are using BasicObject is an indication that they don't want normal method lookup (no methods from Object or Kernel), so I think the assumption that they definitely want normal constant lookup is invalid.

I agree that adding documentation to BasicObject related to this would be beneficial, perhaps you should submit a documentation patch?

#11 - 07/04/2011 12:38 PM - matz (Yukihiro Matsumoto)
- Tracker changed from Bug to Feature

This is not a bug.

#12 - 07/05/2011 02:34 AM - lazaridis.com (Lazaridis Ilias)
=end

Yukihiro Matsumoto wrote:

This is not a bug.

The author has reported a "bug", which was rejected. It's not necessary to change the issue type, in fact it's wrong to do so (because this remains a bug-report and not a feature request, see title).

Some more details on issue-tracking: #4963

=end

#13 - 07/05/2011 08:29 PM - now (Nikolai Weibull)
On Sun, Jul 3, 2011 at 21:05, Thomas Sawyer transfire@gmail.com wrote:

```ruby
ruby-1.9.2-p0 > class X < BasicObject
ruby-1.9.2-p0 ?> include M
```

04/07/2020
Writing

```ruby
include ::M
```

seems to work. Why not use that instead?

#14 - 07/05/2011 11:31 PM - trans (Thomas Sawyer)

@nikolai Yes, that will work in some cases. For a case where it will not, see the eval example I gave above.

#15 - 07/06/2011 12:09 AM - trans (Thomas Sawyer)

@jeremy You make a good case. My general sense of it is YAGNI, but I can't completely rule it out. Who knows, maybe someone will have need of a very clever way to resolve constants for their own classes. But, I think we are perilously close here to that "well-chosen-line" that separates the dependable static language from the convenient dynamic one. If a work around can be found for my eval case, as given above, then I am perfectly happy to concede this issue.

I do want to make one point clear however, as I think your explanation could be interpreted as making a false equivalency. To point, constant look-up and method look-up should not be confused for analogous features. They are in fact quite different. Method look-up operates through the class-hierarchy, while constants are a strange hybrid, which primarily operate through the namespace, but also include the class hierarchy, basically as a matter of convenience. So I still maintain that constant look-up should ultimately terminate at the toplevel (even if BasicObject ignores this). Just as I also am certain that toplevel method definitions really should not be polluting the Object class.

#16 - 07/06/2011 12:34 AM - jeremyevans0 (Jeremy Evans)

Thomas, your example works on 1.9.2p180:

```
$ ruby -v
ruby 1.9.2p180 (2011-02-18 revision 30909) [x86_64-openbsd4.9]
$ cat > q.rb
class R
end
```

```ruby
class Q < BasicObject
def r; R.new; end
def end
$$ irb
  irb(main):001:0> module M; eval(File.read('q.rb')); end
  => nil
  irb(main):002:0> M::Q.new.r
  => #
```

It doesn't work if you change R.new to ::R.new, but that's to be expected (it would be the same if Q descended from Object).

It even works if R and Q are defined in separate files:

```
$ cat > r.rb
class R
end
```

```ruby
$ cat > q.rb
class Q < BasicObject
def r; R.new; end
def end
$$ irb
  irb(main):001:0> module M; eval(File.read('r.rb')); end
  => nil
  irb(main):002:0> module M; eval(File.read('q.rb')); end
  => nil
  irb(main):003:0> M::Q.new.r
  => #
```

The behavior is still the same on "ruby 1.9.3dev (2011-02-28 trunk 30975) [x86_64-openbsd4.9]", so if it no longer works, it must have changed in the last few months.

I agree that constant lookup and method lookup are not the same thing, and should not necessarily be treated the same way. However, I think the purpose of BasicObject is, to the extent possible, remove the default behavior that most objects have in order to allow the programmer to define their own behavior. Therefore, I think allowing the programmer control over constant lookup in BasicObject subclasses makes sense.

#17 - 07/06/2011 04:45 AM - trans (Thomas Sawyer)

You're right, it does work. I recollect testing it, but I must have misconstrued the actual error I was getting at the time. Too long ago now to recall the
Okay. I will write up docs on using `#const_missing` with `BasicObject` and submit it.

Thanks for reviewing this in detail.

#18 - 07/11/2011 12:09 AM - trans (Thomas Sawyer)
I submitted documentation addition.
https://github.com/ruby/ruby/pull/31

#19 - 08/10/2011 12:18 AM - trans (Thomas Sawyer)
Can we merge?

#20 - 08/10/2011 06:10 AM - drbrain (Eric Hodel)
I committed documentation based on your pull request as r32700 in the closed Bug #5067. There is no need to merge.

#21 - 08/10/2011 06:33 AM - trans (Thomas Sawyer)
Okay. That's great.

Reading it over I have a couple of impressions that could help improve upon it:

1) The use of "standard library" is confusing, in contrast to core vs. standard libs.
2) There is no mention of "constant look-up" which would be more technically poignant.
3) The word "like" is a bit over-used.

So when you get a chance maybe you can work these consideration in.

Thanks.