Ruby - Feature #5582

Allow clone of singleton methods on a BasicObject

11/07/2011 12:34 PM - thinkerbot (Simon Chiang)

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

Description

Currently I do not know of a way to implement something like 'clone' on a BasicObject subclass. This is as close as I've gotten but as you can see the singleton methods are not propagated to the clone.

```
require 'test/unit'

class Context < BasicObject
  def _singleton_class_
    class << self
      SINGLETON_CLASS = self
      def _singleton_class_
        SINGLETON_CLASS
      end
    end
    _singleton_class_
  end

  def _class_
    _singleton_class_.superclass
  end

  def _extend_(mod)
    mod.__send__(:extend_object, self)
  end

  def _initialize_clone_(orig)
    # set variables as needed
  end

  def _clone_
    clone = _class_.allocate
    clone._initialize_clone_(self)
    _singleton_class_.included_modules.each {|mod| clone._extend_ mod }
    clone
  end
end

class ContextTest < Test::Unit::TestCase
  module A
    def a
      :a
    end
  end

  def test__clone__inherits_modules
    context = Context.new
    context._extend_ A
    clone = context._clone_
    assert_equal :a, clone.a
  end

  def test__clone__inherits_singleton_methods
    context = Context.new

    def context.a
```

```
      :a
    end

    clone = context._clone_
    assert_equal :a, clone.a  # fails
  end
end
```

Is there a way to do this that I don't see? If not, then I request that a way be added - perhaps by allowing the singleton_class to be set somehow.

In my case I am using Context as the context for a dsl where methods write to a target (an instance variable). I want to be able to clone a context such that I can have multiple contexts with the same methods, including extensions and singletons, that write to different targets.

Thank you.

History

#1 - 11/24/2011 11:34 AM - kernigh (George Koehler)

=begin
My first attempt:

module Clone
include Kernel
(instance_methods - [:clone, :initialize_clone]).each {|m| undef_method m}
end

b = BasicObject.new
class << b
include ::Clone
def single; "Quack!"; end
end

c = b.clone
puts c.single

Output:

scratch.rb:3: warning: undefining `object_id' may cause serious problems
Quack!

Clone inherits from Kernel, but undefines all its instance methods except Clone#clone and Clone#initialize_clone. This technique has some awful side effects: Kernel === b and Kernel === c become true. Clone might inherit metamethods from Kernel (because I only undefined instance methods, not metamethods).
=end

#2 - 03/27/2012 11:52 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

#3 - 11/24/2012 02:14 PM - mame (Yusuke Endoh)

- Target version changed from 1.9.2 to 2.6

#4 - 12/12/2012 11:36 PM - nobu (Nobuyoshi Nakada)

=begin
2.0 allows `method transplanting'.

module Clone
%i[clone initialize_copy initialize_dup initialize_clone].each do |m|
define_method(m, Kernel.instance_method(m))
end
end
=end

#5 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)