Ruby - Feature #8046

allow Object#extend to take a block

03/08/2013 11:48 AM - phluid61 (Matthew Kerwin)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

=begin
In #8038 david_macmahon proposed:

How about allowing Object#extend to take a block that would be executed with the context such that methods defined therein would become singleton methods of the object receiving #extend?

For example:
foo = Object.new
foo.extend do
def bar
# ...
end
def baz
# ...
end
end
=end

History

#1 - 03/08/2013 11:53 AM - Anonymous

=begin
There are two ways to do this - make the extend block execute in the context of the receiver's singleton class, or make it execute in the context of a new module to be mixed in to the receiver's singleton class.

For example:

```
def extend(&bk)
  singleton_class.class_eval(&bk)
end
```

or

```
def extend(&bk)
  singleton_class.send(:include, Module.new(&bk))
end
```

Which should it be?
=end

#2 - 03/08/2013 01:05 PM - phluid61 (Matthew Kerwin)

charliesome (Charlie Somerville) wrote:

> There are two ways to do this - make the extend block execute in the
> context of the receiver's singleton class, or make it execute in the
> context of a new module to be mixed in to the receiver's singleton class.
> ...
> Which should it be?

I'd think more like the former, as that doesn't inject a new anonymous Module into the singleton_class's #ancestors.

Does class_eval do anything dramatically different from module_eval (i.e. is the block handled differently in either case)?

#3 - 03/08/2013 03:53 PM - david_macmahon (David MacMahon)

On Mar 7, 2013, at 6:53 PM, charliesome (Charlie Somerville) wrote:

> There are two ways to do this
>
> ```
> def extend(&bk)
> singleton_class.class_eval(&bk)
> end
> ```
>
> or
>
> ```
> def extend(&bk)
> singleton_class.send(:include, Module.new(&bk))
> end
> ```

At the risk of being overly pedantic, since we're talking about Object#extend, I think it would be more like:

```
def extend(*modules, &bk)
# extend singleton_class with modules, if any
singleton_class.class_eval(&bk) if bk
end
```

or

```
def extend(module=nil, &bk)
# extend singleton_class with modules, if any
singleton_class.send(:include, Module.new(&bk)) if bk
end
```

Which raises another question: what would be the order of extending if #extend is passed one or more modules *and* given a block? IOW, should the passed in module(s) be included first thereby giving the block the opportunity to override them or vice versa (or should this be explicitly disallowed)? I guess I'd favor the first way (include module(s) first, then block can override).

On the original question I tend to agree with @phluid61 that it would be preferable to avoid inserting an anonymous Module in the singleton_class's ancestors. Would having the anonymous module provide any advantage over not having it?

Thanks,
Dave

P.S. Why "singleton_class.send(:include, Module.new(&bk))" instead of just "singleton_class.include(Module.new(&bk))"? Are these somehow not equivalent?

#4 - 03/08/2013 04:27 PM - phluid61 (Matthew Kerwin)

david_macmahon (David MacMahon) wrote:

> Which raises another question: what would be the order of extending if
> #extend is passed one or more modules *and* given a block? IOW, should
> the passed in module(s) be included first thereby giving the block the
> opportunity to override them or vice versa (or should this be explicitly
> disallowed)? I guess I'd favor the first way (include module(s) first,
> then block can override).

That's the order Facets uses: https://github.com/rubyworks/facets/blob/master/lib/core/facets/kernel/extend.rb

> On the original question I tend to agree with @phluid61 that it would be
> preferable to avoid inserting an anonymous Module in the
> singleton_class's ancestors. Would having the anonymous module provide
> any advantage over not having it?

My reasoning against was that on calling it a second time, there would be a second anonymous module, and so on.

> P.S. Why "singleton_class.send(:include, Module.new(&bk))" instead of
> just "singleton_class.include(Module.new(&bk))"? Are these somehow not
> equivalent?

#include is private, so can't be called directly from outside the singleton_class object.

#5 - 03/08/2013 09:19 PM - nobu (Nobuyoshi Nakada)

Without a module, it'd not be #extend but #singleton_class_eval.

#6 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)