

## Ruby trunk - Bug #7248

### Shouldn't Enumerator::Lazy.new be private?

10/31/2012 01:23 PM - marcandre (Marc-Andre Lafortune)

<b>Status:</b> Closed	
<b>Priority:</b> Normal	
<b>Assignee:</b> marcandre (Marc-Andre Lafortune)	
<b>Target version:</b> 2.0.0	
<b>ruby -v:</b> ruby 2.0.0dev (2012-10-29 trunk 37380) [x86_64-darwin10.8.0]	<b>Backport:</b>
<b>Description</b>	
Is there a reason why Enumerator::Lazy.new is not private?	
Lazy enumerators should be created with Enumerable#lazy. Moreover, there is no doc, and it can give unexpected results too.	
<b>Related issues:</b>	
Related to Ruby trunk - Feature #4890: Enumerable#lazy	<b>Closed</b> <b>06/16/2011</b>

#### Associated revisions

##### Revision 44cd5f21 - 02/05/2013 03:49 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Finalize and document Lazy.new. [Bug #7248]  
Add Lazy#to\_enum and simplify Lazy#size.
- test/ruby/test\_lazy\_enumerator.rb: tests for above

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@39057 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

##### Revision 39057 - 02/05/2013 03:49 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Finalize and document Lazy.new. [Bug #7248]  
Add Lazy#to\_enum and simplify Lazy#size.
- test/ruby/test\_lazy\_enumerator.rb: tests for above

##### Revision 39057 - 02/05/2013 03:49 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Finalize and document Lazy.new. [Bug #7248]  
Add Lazy#to\_enum and simplify Lazy#size.
- test/ruby/test\_lazy\_enumerator.rb: tests for above

##### Revision 39057 - 02/05/2013 03:49 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Finalize and document Lazy.new. [Bug #7248]  
Add Lazy#to\_enum and simplify Lazy#size.
- test/ruby/test\_lazy\_enumerator.rb: tests for above

**Revision 39057 - 02/05/2013 03:49 AM - marcandre (Marc-Andre Lafortune)**

- enumerator.c: Finalize and document Lazy.new. [Bug #7248]  
Add Lazy#to\_enum and simplify Lazy#size.
- test/ruby/test\_lazy\_enumerator.rb: tests for above

**Revision 39057 - 02/05/2013 03:49 AM - marcandre (Marc-Andre Lafortune)**

- enumerator.c: Finalize and document Lazy.new. [Bug #7248]  
Add Lazy#to\_enum and simplify Lazy#size.
- test/ruby/test\_lazy\_enumerator.rb: tests for above

**Revision 39057 - 02/05/2013 03:49 AM - marcandre (Marc-Andre Lafortune)**

- enumerator.c: Finalize and document Lazy.new. [Bug #7248]  
Add Lazy#to\_enum and simplify Lazy#size.
- test/ruby/test\_lazy\_enumerator.rb: tests for above

## History

---

**#1 - 11/05/2012 11:01 PM - mame (Yusuke Endoh)**

- Status changed from Open to Assigned
- Assignee set to yhara (Yutaka HARA)

Yhara-san, I'd like your opinion about this ticket.

--

Yusuke Endoh [mame@tsg.ne.jp](mailto:mame@tsg.ne.jp)

**#2 - 12/08/2012 01:17 AM - yhara (Yutaka HARA)**

Hi,

Enumerator::Lazy.new will be needed  
(1) when you want to overwrite behavior of a lazy method. eg:

---

```
class Enumerator::Lazy
  def zip(*args, &block)
    enums = args.map(&:lazy)
    Lazy.new(self){|yielder, val|
      ary = [val] + enums.map{|e| e.next}
      if block
        yielder << block.call(ary) # make lazy.zip{} behave lazy (currently it doesn't because enum.zip{} is eager)
      else
        yielder << ary
      end
    }
  end
end
```

```
fizz = [nil, nil, nil, nil, "Fizz"].cycle
buzz = [nil, nil, "Buzz"].cycle
```

**p fizz.lazy.zip(buzz){|f, b| "#{f}#{b}"}.first(20)**

(2) when you want to add a new Enumerable method and its lazy version. eg:

---

```
module Enumerable
  def filter_map(&block)
    self.map(&block).compact
  end
end
```

```
class Enumerator::Lazy
  def filter_map(*args, &block)
    Lazy.new(self){|yielder, val|
      result = block.call(val)
      yielder << result if result
    }
  end
end
```

```
p [11,12,13].filter_map{|i| i*i if i.even?} #=> [144]
```

## **p (1..Float::INFINITY).lazy.filter\_map{|i| i\*i if i.even?}.first(20)**

**#3 - 12/09/2012 10:44 AM - marcandre (Marc-Andre Lafortune)**

Oh, interesting.

I'll do my best to document it, then.

This leads to more questions, though:

1) Is there a use case for the form without a block?

It's used internally (before calling lazy\_set\_method), but other than that I can't see a good use.

2) Is there a use case for specifying a symbol and arguments?

Again, internally we call lazy\_set\_method, to the symbol and arguments are only used by inspect, right?

3) Is there a good way to improve the inspect of such a lazy enum?

```
p [11,12,13].filter_map{|i| i*i if i.even?} # => #<Enumerator::Lazy: #<Enumerator::Lazy: [1, 2, 3]>:each>
```

Notice the each and no appearance of filter\_map

Doing Lazy.new(self, :filter\_map) does not work and seems redundant.

Thanks

BTW, ultimately, I'm trying to see if Lazy.new can be adapted to accept a size lambda argument...

**#4 - 01/12/2013 04:58 AM - yhara (Yutaka HARA)**

*- Assignee changed from yhara (Yutaka HARA) to marcandre (Marc-Andre Lafortune)*

marcandre (Marc-Andre Lafortune) wrote:

Oh, interesting.

I'll do my best to document it, then.

Thanks!

This leads to more questions, though:

1) Is there a use case for the form without a block?

It's used internally (before calling lazy\_set\_method), but other than that I can't see a good use.

That form is only for internal use.

You can remove the form without a block by replacing Lazy.new(enum)' with Lazy.new(enum){|y, v| y<<v}'.

2) Is there a use case for specifying a symbol and arguments?

Actually I did not know lazy\_initialize can take a symbol :-p  
So I'm not sure about how the symbol and arguments are used,  
but it looks like for internal use.  
According to svn annotate, it is introduced to implement lazy.cycle ([r35028](https://r35028)).

Again, internally we call lazy\_set\_method, to the symbol and arguments are only used by inspect, right?

That seems right.

3) Is there a good way to improve the inspect of such a lazy enum?

```
p [11,12,13].filter_map{|i| i*i if i.even?} # => #<Enumerator::Lazy: #<Enumerator::Lazy: [1, 2, 3]>:each>
```

Notice the each and no appearance of filter\_map

Doing Lazy.new(self, :filter\_map) does not work and seems redundant.

BTW, ultimately, I'm trying to see if Lazy.new can be adapted to accept a size lambda argument...

Well, I have no idea. It would be difficult to design Lazy.new which may take  
obj, block, symbol, args and size/size\_fn...

BTW, I have a question. Document of to\_enum says "see Enumerator#size=" but there is no such method. Is it a typo?  
<https://github.com/ruby/ruby/blob/e90ccd3beb0b9bf1125461ef68943578739bebbe/enumerator.c#L201>

#### #5 - 01/14/2013 01:59 PM - marcandre (Marc-Andre Lafortune)

- Priority changed from Normal to 5

Thanks for the answers.

So, the public API of Lazy.new has the following issues:

- not documented
- should require a block but doesn't
- accepts a method name & arguments which aren't really usable
- has a misleading "inspect"

Moreover, the only way for the user to create a lazy enumerator with a size is to subclass Lazy.

Here's what I propose as the official Lazy.new documentation and API:

```
Lazy.new(obj, size=nil) { |yielder, *values| ... }
```

Creates a new Lazy enumerator. When the enumerator is actually enumerated (e.g. by calling #force), +obj+ will be enumerated and each value passed to the given block. The block can yield values back using +yielder+. For example, to create a method +filter\_map+ in both lazy and non-lazy fashions:

```
module Enumerable
  def filter_map(&block)
    map(&block).compact
  end
end
```

```
class Enumerator::Lazy
  def filter_map
    Lazy.new(self) do |yielder, *values|
      result = yield *values
      yielder << result if result
    end
  end
end
```

```
(1..Float::INFINITY).lazy.filter_map{|i| i*i if i.even?}.first(5)
# => [4, 16, 36, 64, 100]
```

Does this seem acceptable to you?

We should also change the result of the 'inspect' method for these user created lazy enumerators to remove the 'each', i.e:

```
(1..Float::INFINITY).lazy.filter_map{|i| i*i if i.even?}.inspect
# => #<Enumerator::Lazy: #<Enumerator::Lazy: 1..Infinity>>
```

If we want to provide an easy way to provide more info in the inspect, we could add extra parameters, but they shouldn't be used when iterating, only for inspection...

BTW, I have a question. Document of `to_enum` says "see `Enumerator#size`=" but there is no such method. Is it a typo?

Typo fixed, thanks.

#### #6 - 01/14/2013 01:59 PM - marcandre (Marc-Andre Lafortune)

- Assignee changed from marcandre (Marc-Andre Lafortune) to yhara (Yutaka HARA)

#### #7 - 01/17/2013 12:10 AM - yhara (Yutaka HARA)

- Assignee changed from yhara (Yutaka HARA) to marcandre (Marc-Andre Lafortune)

marcandre (Marc-Andre Lafortune) wrote:

Here's what I propose as the official `Lazy.new` documentation and API:

```
Lazy.new(obj, size=nil) { |yielder, *values| ... }
```

Creates a new `Lazy` enumerator. When the enumerator is actually enumerated (e.g. by calling `#force`), `+obj+` will be enumerated and each value passed to the given block. The block can yield values back using `+yielder+`. For example, to create a method `+filter_map+` in both `lazy` and `non-lazy` fashions:

```
module Enumerable
  def filter_map(&block)
    map(&block).compact
  end
end
```

```
class Enumerator::Lazy
  def filter_map
    Lazy.new(self) do |yielder, *values|
      result = yield *values
      yielder << result if result
    end
  end
end
```

```
(1..Float::INFINITY).lazy.filter_map{|i| i*i if i.even?}.first(5)
# => [4, 16, 36, 64, 100]
```

Does this seem acceptable to you?

Yes!

#### #8 - 02/05/2013 12:49 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset [r39057](#).

Marc-Andre, thank you for reporting this issue.

Your contribution to Ruby is greatly appreciated.

May Ruby be with you.

- 
- `enumerator.c`: Finalize and document `Lazy.new`. [Bug [#7248](#)]  
Add `Lazy#to_enum` and simplify `Lazy#size`.
  - `test/ruby/test_lazy_enumerator.rb`: tests for above