# Ruby master - Feature #666

## Enumerable::to_hash

10/20/2008 02:25 PM - marcandre (Marc-Andre Lafortune)

| | | |
|---|---|---|
| **Status:** | Rejected | |
| **Priority:** | Normal | |
| **Assignee:** | matz (Yukihiro Matsumoto) | |
| **Target version:** | 2.0.0 | |

**Description**

=begin
There are many ways to obtain an array from enumerables (to_a, map, ...).
There is no natural way to obtain a hash from an enumerable (except for Hash[some_array]).
There is a Hash::to_a but no Array::to_hash.
Here is what I would like:

[:hello, "world"], [:choice, [:red_pill, :blue_pill]].to_hash ==> {:hello=>"world", :choice=>[:red_pill, :blue_pill]}
(1..3).to_hash{|n| [n, n**2]} ==> {1 => 1, 2 ==> 4, 3 ==> 9}

I propose to add the following Enumerable::to_hash :

module Enumerable
def to_hash
result = {}
self.each do |key, value|
key, value = yield(key, value) if block_given?
result[key] = value
end
result
end
end

Since Hash::to_a returns an array of key-value pairs, I fell it's natural that a block to construct a Hash should return key-value pairs.
This definition has nice symmetric properties: for any Hash h, the following all return a copy of h.
h.to_a.to_hash
h.to_hash{|p| p}
h.to_hash{|k,v| [k,v]}
h.keys.zip(h.values).to_hash

Thank you for your attention,

Marc-Andre Lafortune
=end

**Related issues:**

| | | |
|---|---|---|
| Related to Ruby master - Feature #4151: Enumerable#categorize | **Rejected** | |
| Related to Ruby master - Feature #7292: Enumerable#to_h | **Closed** | **11/07/2012** |
| Has duplicate Ruby master - Feature #7241: Enumerable#to_h proposal | **Rejected** | **10/30/2012** |

### History

**#1 - 11/29/2008 04:27 PM - ko1 (Koichi Sasada)**

*- Assignee set to matz (Yukihiro Matsumoto)*

=begin

=end

**#2 - 12/11/2008 12:07 PM - yugui (Yuki Sonoda)**

*- Target version set to 2.0.0*

=begin

=end

#### #3 - 04/17/2009 01:51 PM - marcandre (Marc-Andre Lafortune)

=begin
Anyone eagerly waiting for this feature will be interested to read http://redmine.ruby-lang.org/issues/show/1385
=end

#### #4 - 04/19/2009 04:07 AM - matz (Yukihiro Matsumoto)

*- Status changed from Open to Rejected*

=begin
Enumerable in general does not correspond with mappings, so that I feel Enumerable#to_hash is improper.

=end

#### #5 - 05/12/2009 03:43 PM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:23298] Re: Feature #666 Enumerable::to_hash"
on Fri, 24 Apr 2009 00:08:53 +0900, Marc-Andre Lafortune ruby-core-mailing-list@marc-andre.ca writes:
|
|On Thu, Apr 23, 2009 at 9:55 AM, Michael Fellinger
|m.fellinger@gmail.com wrote:
|
|> Doesn't the new behaviour of Hash::[] solve these cases just as well?
|
|Yes indeed it does, but
|
|1) The new form of Hash[] has yet to be confirmed by Matz (see
|http://redmine.ruby-lang.org/issues/show/1385 ).

Didn't I? I confirm.

|2) It's not as natural as #to_hash. Don't we usually use instance
|methods to convert between types? If you look at conversion between
|basic types, you can convert:
|Numeric <=> String <=> Symbol
|Hash => Array
|All these using instance methods. The only arrow missing is from Array
|back to Hash!

Even though a hash can be represented by an array, there's not always
natural map from Array to Hash. I am not sure how much to_hash is
useful, when we cannot define what [1,2,3].to_hash should return.

```
                          matz.
```

=end

#### #6 - 03/24/2011 05:35 AM - tokland (Arnau Sanchez)

=begin
Hi,

I don't know if it's polite to comment in old closed issues, excuse me if it's not.

I have to say that I wholeheartedly agree with Marc-Andre: the lack of Enumerable-to-Hash conversion is important; in my experience it's an extraordinarily common transformation. Let's see what people usually does (unaware of Facet's Enumerable#mash):

1) novice way

```
h = {}
(1..3).each { |n| h[n] = n**2 }
h
```

This is just ugly compared with the beautiful, compact, functional code we usually write in Ruby. Moreover, being imperative, it cannot be used in a expression.

2) Hash:

```
Hash[(1..3).map { |n| [n, n**2] }]
```

Not bad, but it's disappointing in a OOP language to "go back", you'd expect to write from left-to-right as usual and use a method. Moreover, it's less efficient because it needs an intermediate array to be built.

3) Enumerable#inject (+update/merge).

(1..3).inject({}) { |hash, n| hash.update(n => 2*n) }

Too verbose, the intent is hidden by the infrastructure.

I think we all agree nothing is clearer than (mash or whatever name):

(1..3).mash { |n| [n, 2*n] }

Finally, answering to Matz prevention:

> we cannot define what [1,2,3].to_hash should return

Somehow it's already defined:

```
Hash1,2,3
=> {}
```

Although it would be also ok to raise an exception (as Python does, for example). A mapping has been always represented by a collection of pairs (key, value), all languages with minimal functional capabilities (and Ruby has powerful ones) has such function/method transformation.
=end

#### #7 - 06/09/2011 11:03 PM - mfn (Markus Fischer)

Arnau Sanchez wrote:

> I don't know if it's polite to comment in old closed issues, excuse me if it's not.
>
> I have to say that I wholeheartedly agree with Marc-Andre: the lack of Enumerable-to-Hash conversion is important; in my experience it's an extraordinarily common transformation. Let's see what people usually does (unaware of Facet's Enumerable#mash):
> [...]
> Hash[(1..3).map { |n| [n, n**2] }]
>
> Not bad, but it's disappointing in a OOP language to "go back", you'd expect to write from left-to-right as usual and use a method. Moreover, it's less efficient because it needs an intermediate array to be built.
>
> Somehow it's already defined:
>
> ```
> Hash1,2,3
> => {}
> ```
>
> Although it would be also ok to raise an exception (as Python does, for example). A mapping has been always represented by a collection of pairs (key, value), all languages with minimal functional capabilities (and Ruby has powerful ones) has such function/method transformation.

I was about to open a new feature request when I found this, unfortunately rejected, issue.

I'd also love to see Hash[] being available as Array#to_h too; it's just much more convenient. I recently had the urge to sort a hash and would could have been:

some_hash.sort { |a,b| whatever_is_necessary }.to_h

had to be

Hash[ some_hash.sort { |a,b| whatever_is_necessary } ]

- Markus

#### #8 - 06/10/2011 03:26 AM - marcandre (Marc-Andre Lafortune)

Thanks for commenting on this old request.

You might want to read the thread [ruby-core:33683] on Akira's proposal for Enumerable#categorize and my alternative proposal Enumerable#associate which would act as a more versatile Enumerable#to_hash.

Your input could have more impact on that thread than on this one. Hopefully we can come up with a neat functionality for the some future version of Ruby.

#### #9 - 06/10/2011 05:23 AM - mfn (Markus Fischer)

Hi,

On 09.06.2011 20:26, Marc-Andre Lafortune wrote:

> You might want to read the thread [ruby-core:33683] on Akira's proposal for Enumerable#categorize and my alternative proposal Enumerable#associate which would act as a more versatile Enumerable#to_hash.
>
> Your input could have more impact on that thread than on this one. Hopefully we can come up with a neat functionality for the some future version of Ruby.

Thanks for the pointer, very informative. I choose not to add anything to the other thread, as it seems they goal is a bit different.

My one and only intention is really simple: provide the reverse of Hash#to_a ("Converts hsh to a nested array of [ key, value ] arrays.") ; e.g. Array#to_h .

I understood from the other thread much more flexible solutions where sought, nothing I could aid anything valuable I fear.

I'm just a novice when it comes to Ruby and found a frequent need for that functionality; maybe it's because of my non-Ruby background and thus my non-Ruby approach. Likely also that it's not as simple as I wished this could be, so far Hash[ ... ] was always the solution for me so

class Array ; def to_h ; Hash[ self ]; end; end

worked very well for me.

cheers,

- Markus

#### #10 - 02/13/2014 10:38 AM - tokland (Arnau Sanchez)

For those interested in this feature, check #7292, Marc-Andre implemented Array#to_h and Enumerable#to_h. It's not as powerful (since it takes no block, you'll usually need to create an intermediate array with "map"), but it's definitely better than Hash[pairs]. Thank you Marc-Andre!