Ruby - Bug #3128

Randomness specs

04/11/2010 03:34 PM - marcandre (Marc-Andre Lafortune)

Status:	Closed		
Priority:	Normal		
Assignee:	matz (Yukihiro Matsumoto)		
Target version:			
ruby -v:		**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN

Description

=begin
What should be the Ruby specs for the new Random class (and existing Kernel.{s}rand)?

More precisely: what should one expect of any Ruby implementation?

Several degrees of similarity with MRI are possible:

Say r = Random.new(42) and N is an Integer

1. r.rand(N) is included in 0...N
2. r.rand(N) will eventually return all values in 0...N
3. r.rand(N) will return any particular value in 0...N with a probability of around 1/N
   <insert any any of the known random tests and/or a minimal period>
4. r is a Mersenne Twister
5. r is MT19937
6. r.rand(N) generates the same particular string on all platforms

Implementers are ultimately free to choose whichever implementation they prefer, of course.

Still, it would be preferable to state what is the expect behavior for the Ruby language, not just for MRI. If MRI's guarantees are stronger, these should be stated as such.

Current state of affairs:

The documentation for Random class states that it is a Mersenne Twister pseudo number generator (but doesn't state which), so this corresponds to level (3) above.

Kernel.rand states that *currently*, r is a modified MT19937.

The Ruby Standardization WG Draft doesn't document Kernel.rand yet (nor Random, of course).

Rubinius' implementation currently guarantees level 0 (and also level 1 if N is not too big)

JRuby's implementation guarantees level 1 (and also level 2 if N is not too big)

Python and Java both guarantee the same particular sequence; other algorithms might be available as subclasses of their Random class.

My personal choice would be in line with Java and Python: insure the exact same sequence for the Random class. Implementations are free to provide subclasses of Random for different/better/faster algorithms.

If deemed preferable, Kernel.rand could have much lower required standards to allow for better speed, in which case level (2) above seems like the strict minimum.
=end

History

#1 - 04/11/2010 06:02 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Sun, 11 Apr 2010 15:34:07 +0900,
Marc-Andre Lafortune wrote in [ruby-core:29447]:

> Say r = Random.new(42) and N is an Integer
>
> 1. r.rand(N) is included in 0...N
> 2. r.rand(N) will eventually return all values in 0...N
> 3. r.rand(N) will return any particular value in 0...N with a probability of around 1/N
>   <insert any any of the known random tests and/or a minimal period>
> 4. r is a Mersenne Twister
> 5. r is MT19937
> 6. r.rand(N) generates the same particular string on all platforms

I think the specs are only 0 and 5, and I'd categorize them as:

A) specs/restrictions:

1. r.rand(N) SHOULD NOT return a value which is NOT included in 0...N
2. Random class instances which are created with same initial seed values SHOULD generate same values in each calls, on all platforms

B) degrees of randomness:

1. r.rand(N) will eventually return all values in 0...N
2. r.rand(N) will return any particular value in 0...N with a probability of around 1/N
3. other random tests

C) implementation details:

1. r is a modified MT19937 (in the current MRI)
2. implementers are ultimately free to choose whichever implementation they prefer

--
Nobu Nakada

=end

#2 - 04/13/2010 12:49 PM - mame (Yusuke Endoh)

=begin
Hi,

2010/4/11 Marc-Andre Lafortune redmine@ruby-lang.org:

> 1. r.rand(N) will eventually return all values in 0...N

I could be paranoid, but I doubt whether it can be guaranteed currently. This is because there is Bignum in Ruby.

Pseudorandom numbers generator (even Mersenne Twister) creates long, but finite cycle of sequence of numbers. IOW, eventually the sequence repeats.
So, when N is bigger than the period (cycle length), there is a number (in 0..N) that cannot be returned from r.rand(N).
It is Pigeonhole principle.

If there is a PRNG whose period can be configurable, it can be solved. But as far as I know, the period of MT19937 is fixed amount of 2**19937-1.

Of course, it is not problematic in practice. But I guess it should not be declared as the spec.

Just my two cents,

--
Yusuke Endoh mame@tsg.ne.jp

=end

#3 - 06/26/2011 04:00 PM - nahi (Hiroshi Nakamura)

- Status changed from Open to Assigned

- Assignee set to nahi (Hiroshi Nakamura)

- Target version changed from 1.9.2 to 1.9.3

#4 - 06/28/2011 10:04 PM - nahi (Hiroshi Nakamura)

- Status changed from Assigned to Open

- Assignee deleted (nahi (Hiroshi Nakamura))

- Target version deleted (1.9.3)

It's a Ruby language specification issue, so I clear 'Target version'.

#5 - 04/17/2012 08:30 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to marcandre (Marc-Andre Lafortune)

Marc-Andre, do you need discussion about this?

--
Yusuke Endoh mame@tsg.ne.jp

#6 - 04/30/2012 11:47 AM - marcandre (Marc-Andre Lafortune)

- Assignee changed from marcandre (Marc-Andre Lafortune) to matz (Yukihiro Matsumoto)

- Priority changed from Normal to 3

Hi,

mame (Yusuke Endoh) wrote:

> Marc-Andre, do you need discussion about this?

After your remarks and those of Nobu, the question becomes:

Should Randomg.new(42); rand return the same value in all Ruby implementations, or is the result implementation defined?

I'm assigning this to Matz.

#7 - 04/18/2013 05:59 AM - naruse (Yui NARUSE)

- Tracker changed from Misc to Bug

#8 - 11/22/2017 10:08 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Assigned to Closed