Ruby master - Feature #908

Should be an easy way of reading N characters from am I/O stream

12/20/2008 05:54 AM - mike (Michael Selig)

Status:	Rejected	
Priority:	Normal	
Assignee:	akr (Akira Tanaka)	
Target version:	2.6	

Description

=begin
I know of no way of reading N characters (as opposed to bytes) and returning a string other than writing a loop with getc or each_char.

Suggestions - one of:

1) Change the "limit" paramater on IO#gets to mean characters rather than bytes. This would change existing behaviour, so may be unacceptable.

2) Add an optional "limit" parameter to IO#getc, so IO#getc will read up to this many characters
=end

History

#1 - 02/03/2009 10:42 AM - shyouhei (Shyouhei Urabe)

- Assignee set to akr (Akira Tanaka)

=begin

=end

#2 - 02/03/2009 10:44 AM - akr (Akira Tanaka)

=begin
What's the usecases?
=end

#3 - 02/03/2009 10:21 PM - radarek (Radosław Bułat)

=begin
Reading N characters from stream is (at least for me) as natural as reading N bytes. Usecases are almost the same as for bytes but when you want operate on characters. For example you have file encoded in utf-8 and want to read first 10 characters.
=end

#4 - 02/03/2009 11:06 PM - JEG2 (James Gray)

=begin
I needed this in the standard CSV library.

My use case was that I peek ahead in the stream to determine what kind of line endings it has. I just grab a block of characters and see if I find any standard line endings in there. This was pretty challenging in Ruby 1.9, because just reading some bytes meant I had great chances to have picked up invalid data. Then, when I hit it with a Regexp to find the line endings, an Exception is raised.

I'm using this code to get around that problem in CSV:

```
#
# Reads at least +bytes+ from @io, but will read up 10 bytes ahead if
# needed to ensure the data read is valid in the ecoding of that data. This
# should ensure that it is safe to use regular expressions on the read data,
# unless it is actually a broken encoding. The read data will be returned in
# @encoding.
#
def read_to_char(bytes)
return "" if @io.eof?
data = @io.read(bytes)
begin
encoded = encode_str(data)
raise unless encoded.valid_encoding?
```

```
return encoded
rescue  # encoding error or my invalid data raise
if @io.eof? or data.size >= bytes + 10
return data
else
data += @io.read(1) until data.valid_encoding? or
@io.eof?           or
data.size >= bytes + 10
retry
end
end
end
```

That worked for CSV, where I just need some characters and don't have to have an exact count. If you do need an exact count though, the code gets more complicated.

I agree that this is something Ruby should do for us.

=end

#5 - 02/04/2009 12:47 AM - radarek (Radosław Bułat)

=begin
I wonder also about byte-oriented IO#seek if someone want to have
character-oriented seek. It looks like byte-oriented seekd is useless
in multibyte character-oriented stream because it could jump to bad
position (in the middle of character bytes).

--
Pozdrawiam

Radosław Bułat
http://radarek.jogger.pl - mój blog

=end

#6 - 02/04/2009 08:27 AM - mike (Michael Selig)

=begin
I have a simple use-case:

Existing datafile has fixed length records, currently single-byte chars. I want to convert the application (which is quite old) to support multi-byte characters, but I don't want to have to go to the trouble of changing to variable-length or delimited records/fields. I would like to be able to read each record (whose length in chars I know) with one operation, instead of looping through each character.

Re: seeking by character count - it would be nice, but I have no idea how it could be implemented efficiently!
=end

#7 - 02/04/2009 09:34 AM - hramrach (Michal Suchanek)

=begin
On 04/02/2009, Michael Selig redmine@ruby-lang.org wrote:

> Re: seeking by character count - it would be nice, but I have no idea how it could be implemented efficiently!

For UTF-8/UTF-16/SJIS/EUC-JP/BIG5 .. no. UTF-32 is dword aligned but
you cannot tell what byte ordering it uses reliably. Bad thing. Seeks
are probably not for text files or only for text files you have parsed
already so you know where you are seeking.

Thanks

Michal

=end

#8 - 02/04/2009 09:38 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:21817] Re: [Feature #908] Should be an easy way of reading N characters from am I/O stream"
on Wed, 4 Feb 2009 09:33:44 +0900, Michal Suchanek hramrach@centrum.cz writes:

|> Re: seeking by character count - it would be nice, but I have no idea how it could be implemented efficiently!
|
|For UTF-8/UTF-16/SJIS/EUC-JP/BIG5 .. no. UTF-32 is dword aligned but
|you cannot tell what byte ordering it uses reliably. Bad thing. Seeks
|are probably not for text files or only for text files you have parsed
|already so you know where you are seeking.

Right. Hence I reject the character based seek. Thank you.

Regarding the original N character read, I am positive, but still
haven't decided yet for API.

```
                                matz.
```

=end

#9 - 02/15/2009 10:11 AM - hramrach (Michal Suchanek)

=begin
2009/2/14 Tanaka Akira akr@fsij.org:

> In article op.uotab6oa9245dp@kool,
> "Michael Selig" michael.selig@fs.com.au writes:
>
>> That's right - These files are quite small, and I only need to do
>> sequential I/O. I want to keep the format backward compatible when using a
>> single-byte encoding.
>
> Whould you show an example of such format?
>
> I couldn't imagine a fixed length field which single byte
> encoding (US-ASCII) is usable and multibyte encoding is
> useful.
>
> For example, zip code or some fixed numbering system is
> fixed length but multibyte encoding is not useful.

Let's make it more general - what about the first N characters or first N lines?

I'm sure you can understand this is useful.

How does the lines() Enumerator interact with the IO?

If a method like head(N) was implemented on it would it leave the IO
pointing to the text after the first N records, be it chars, lines, or
anything else?

Can that Enumerator be created so that it starts enumerating at the
current file position?

Thanks

Michal

=end

#10 - 02/16/2009 07:53 PM - hramrach (Michal Suchanek)

=begin
2009/2/15 Tanaka Akira akr@fsij.org:

> In article a5d587fb0902141711q780f0d24jef9be9b8bbe69b2a@mail.gmail.com,
> Michal Suchanek hramrach@centrum.cz writes:
>
>>> For example, zip code or some fixed numbering system is
>>> fixed length but multibyte encoding is not useful.
>>
>> Let's make it more general - what about the first N characters or first N lines?
>>
>> I'm sure you can understand this is useful.

I think I don't understand the usefulness until an actual
example is shown.

If a method like head(N) was implemented on it would it leave the IO
pointing to the text after the first N records, be it chars, lines, or
anything else?

What is represented by the N chars?

I don't understand the question. N chars are N chars, they do not
represent anything else.

It's actually not that hard except the synchronization is not perfect.
By using chars and then lines I lost "F"

```
irb(main):001:0> f=File.open "rom.asm"
=> #File:rom.asm
irb(main):002:0> f.chars.take(10)
=> ["0", "0", "0", "0", "0", "0", "0", "0", " ", " "]
irb(main):003:0> f.lines.take(3)
=> ["A               cli\n", "00000001  FC               cld\n",
"00000002  66670F0115000000  o32 lgdt [dword 0x0]\n"]
irb(main):004:0> f.seek(0)
=> 0
irb(main):005:0> f.lines.take(1)
=> ["00000000  FA               cli\n"]
```

Thanks

Michal

=end

#11 - 02/19/2009 12:21 AM - hramrach (Michal Suchanek)

=begin
2009/2/18 Tanaka Akira akr@fsij.org:

In article a5d587fb0902160252u56b50cfdv8e0fd36bb4f0b1b3@mail.gmail.com,
Michal Suchanek hramrach@centrum.cz writes:

What is represented by the N chars?

I don't understand the question. N chars are N chars, they do not
represent anything else.

I expect something like person's name, zip code, etc.

However, person's name is variable length.

The zip code (in Japan) is fixed length but multibyte
encoding is not useful because it uses only digits.

As was explained by the original poster there are file formats similar
to CSV that use fixed field length instead of separators. I have
myself used such files, and they were in 8-bit fixed width encoding.

However, if you want to "upgrade" your code that uses such files to
multibyte for international support you need reading N characters.

Of course, the alternative is to change your code to use a different
format.This might make exports to and imports from legacy applications
hard, however.

Sure, the export can never be perfect if the files really contain
internationalized data because recoding to the legacy format and
encoding loses some information then.

I'm not sure the usage of the method for "reading N
characters".

Yes, reading N characters does not seem very useful outside of very
specialized scenarios. Most sane file formats use string length in
bytes or separators.

However, reading N characters, lines, or any other units for which you
have an IO enumerator seems useful to me.

Actually reading N lines using the correct line separator would fetch
N records from the file without the need to construct a loop for that
(or repeat the method for reading a line N times).

> > It's actually not that hard except the synchronization is not perfect.
> > By using chars and then lines I lost "F"
>
> I guess enumerator uses lookahead.

That's unfortunate for using the Enumerator with other methods for
reading files.

Thanks

Michal

=end

#12 - 02/19/2009 07:55 PM - hramrach (Michal Suchanek)

=begin
2009/2/19 Tanaka Akira akr@fsij.org:

> In article op.upklh9q19245dp@kool,
> "Michael Selig" michael.selig@fs.com.au writes:
>
> > Also it seems to me that the current usage of the "limit" parameter of
> > IO#gets is not intuitive in 1.9. It is "maximum number of bytes, but don't
> > split a character", and I think it should be changed to mean "maximum
> > number of chars". That would be much more obvious, more useful (IMHO), and
> > still be backward compatible with 1.8.
>
> It is introduced for security reason. bytes are more stable
> than characters.

However, the security would be served as well by a character limit.

As I understand it this limit is introduced so that a gets does not
read several gigabytes of data at once in case there is no line
separator.

Thanks

Michal

=end

#13 - 02/19/2009 08:01 PM - hramrach (Michal Suchanek)

=begin
2009/2/19 Tanaka Akira akr@fsij.org:

> In article op.upklh9q19245dp@kool,
> "Michael Selig" michael.selig@fs.com.au writes:
>
> > In more detail: I have a legacy system that uses fixed length fields. Yes,
> > a name is variable length, but some old systems use a fixed length field,
> > say 40 chars, which is space filled on the right (or truncated). In my
> > case, the data input is by a form, and each field is fixed width. I am
> > changing the system so that the SAME forms can be used, but extended to
> > use UTF-8 not just ASCII. So this means that the number of characters is
> > still fixed, but the number of bytes is no longer fixed. I do *not* want
> > to change the format of the file (though it probably should be, but that

> > would be a lot more work), because I want the application to be backward
> > compatible (when using ASCII data).
>
> This is what I'd like to hear. Thank you for explanation.
>
> It seems the number, 40, is a number for "big enough for
> names".
>
> Why don't you use 40 bytes data format, both with Ruby 1.8
> and 1.9?
>
> Do you think that 40 bytes is not big enough for names in
> some country?
>
> If the data format uses 40 bytes, instead of 40 chars,
> it is easy to read it in Ruby 1.8, even if it contains UTF-8
> chars.

While this might ease working with the file data it might make
designing the form more challenging.

The things to consider:

- checking for byte length rather than character length in something like JavaScript (probably possible)
- explaining the length limit to the user of the application (I would not want to do that)
- making sure that 40 bytes is long enough for names in languages that use exotic characters

Thanks

Michal

=end

#14 - 02/20/2009 01:51 PM - duerst (Martin Dürst)

=begin
At 19:59 09/02/19, Michael Selig wrote:

> Also there are reports reading the data which expect the data to be 40
>
> characters wide. If it wasn't 40 chars, the formatting of the report may
>
> screw up.

Hello Michael,

In general, I agree that being able to work with character numbers
is desirable. The implementation isn't exactly easy, but I hope
eventually we will get there. My current guess is that this might
mean that we have to move IO and related stuff a bit more towards
a model with classes stacked on top of each other. But that's just
a guess.

But regarding your point of format screwup, measuring things in
characters won't help. Assuming that each character has the same
width just doesn't carry very far if you look at all the scripts
around the world.

Regards,   Martin.

#-#-# Martin J. Du"rst, Assoc. Professor, Aoyama Gakuin University
#-#-# http://www.sw.it.aoyama.ac.jp   mailto:duerst@it.aoyama.ac.jp

=end

#15 - 02/20/2009 01:51 PM - duerst (Martin Dürst)

=begin
At 19:00 09/02/19, Tanaka Akira wrote:

> It seems the number, 40, is a number for "big enough for
> names".

> Why don't you use 40 bytes data format, both with Ruby 1.8
> and 1.9?
>
> Do you think that 40 bytes is not big enough for names in
> some country?

Very much so. A typical example would be Georgia, where
many names are as long as some of the longer ones in
Europe, but they require 3 bytes per character.

>> Also it seems to me that the current usage of the "limit" parameter of
>>
>> IO#gets is not intuitive in 1.9. It is "maximum number of bytes, but don't
>>
>> split a character", and I think it should be changed to mean "maximum
>>
>> number of chars". That would be much more obvious, more useful (IMHO), and
>>
>> still be backward compatible with 1.8.

> It is introduced for security reason. bytes are more stable
> than characters.

Can you give more specific explanations of why reading a number
of characters might not be secure?

Regards,   Martin.

#-#-# Martin J. Du"rst, Assoc. Professor, Aoyama Gakuin University
#-#-# http://www.sw.it.aoyama.ac.jp   mailto:duerst@it.aoyama.ac.jp

=end

#16 - 02/23/2009 05:34 PM - duerst (Martin Dürst)

=begin
At 01:00 09/02/23, Tanaka Akira wrote:

> In article 6.0.0.20.2.20090220134502.0823ee98@localhost,
> Martin Duerst duerst@it.aoyama.ac.jp writes:
>
>> Can you give more specific explanations of why reading a number
>> of characters might not be secure?
>
> I considered ISO-2022-JP, Unicode combining characters and
> Punycode.
>
> In these encodings, fixed number of characters doesn't limit
> the number of bytes.

Why do you think there is a need to limit the number of bytes?
In general, that's not how Ruby works, at least not as far as
I understand.

Regards,   Martin.

> However they may not cause problem now because Ruby doesn't
> support combining characters, etc. But Ruby's encoding
> system is extensible. It is possible to define an encoding
>
> ## which makes the character-wise limit insecure.
>
> Tanaka Akira

#-#-# Martin J. Du"rst, Assoc. Professor, Aoyama Gakuin University
#-#-# http://www.sw.it.aoyama.ac.jp   mailto:duerst@it.aoyama.ac.jp

=end

#17 - 02/23/2009 07:02 PM - hramrach (Michal Suchanek)

=begin
2009/2/22 Michael Selig michael.selig@fs.com.au:

> On Mon, 23 Feb 2009 03:00:41 +1100, Tanaka Akira akr@fsij.org wrote:
>
>> In article 6.0.0.20.2.20090220134502.0823ee98@localhost,
>> Martin Duerst duerst@it.aoyama.ac.jp writes:
>>
>>> Can you give more specific explanations of why reading a number
>>> of characters might not be secure?
>>
>> I considered ISO-2022-JP, Unicode combining characters and
>> Punycode.
>>
>> In these encodings, fixed number of characters doesn't limit
>> the number of bytes.
>
> Sure, but how does that make it "insecure"?
>
>> However they may not cause problem now because Ruby doesn't
>> support combining characters, etc. But Ruby's encoding
>> system is extensible. It is possible to define an encoding
>> which makes the character-wise limit insecure.
>
> Sorry, I do not really understand what you mean by the word "insecure".
> Perhaps you could explain what you mean in more detail.
> Also I still do not understand why you say the character limit might be
> "insecure". Can you give an example, please?

Theoretically if separate (combining) character accents are considered
part of the character then a character might be quite long - I guess
about ten codepoints which can be themselves up to six bytes. However,
the number of accents one can put together should be limited to
meaningful combinations so this should still be secure - as long as
the code which determines what is a valid character does not have
bugs. This might be tricky in some cases, though.

Thanks

Michal

=end

#18 - 02/23/2009 07:36 PM - hramrach (Michal Suchanek)

=begin
2009/2/23 Michal Suchanek hramrach@centrum.cz:

> 2009/2/22 Michael Selig michael.selig@fs.com.au:
>
>> On Mon, 23 Feb 2009 03:00:41 +1100, Tanaka Akira akr@fsij.org wrote:
>>
>>> In article 6.0.0.20.2.20090220134502.0823ee98@localhost,
>>> Martin Duerst duerst@it.aoyama.ac.jp writes:
>>>
>>>> Can you give more specific explanations of why reading a number
>>>> of characters might not be secure?
>>>
>>> I considered ISO-2022-JP, Unicode combining characters and
>>> Punycode.
>>>
>>> In these encodings, fixed number of characters doesn't limit
>>> the number of bytes.
>>
>> Sure, but how does that make it "insecure"?
>>
>>> However they may not cause problem now because Ruby doesn't

> > > support combining characters, etc. But Ruby's encoding
> > > system is extensible. It is possible to define an encoding
> > > which makes the character-wise limit insecure.
> >
> > Sorry, I do not really understand what you mean by the word "insecure".
> > Perhaps you could explain what you mean in more detail.
> > Also I still do not understand why you say the character limit might be
> > "insecure". Can you give an example, please?
>
> Theoretically if separate (combining) character accents are considered
> part of the character then a character might be quite long - I guess
> about ten codepoints which can be themselves up to six bytes. However,
> the number of accents one can put together should be limited to
> meaningful combinations so this should still be secure - as long as
> the code which determines what is a valid character does not have
> bugs. This might be tricky in some cases, though.

BTW the same goes for "reading N bytes up to a character boundary"
unless you are willing to accept that you might read nothing even if
data is available because the character did not fit into N bytes, even
for a large N.

Thanks

Michal

=end

#19 - 02/24/2009 10:07 AM - duerst (Martin Dürst)

=begin
At 19:02 09/02/23, Michal Suchanek wrote:

> Theoretically if separate (combining) character accents are considered
> part of the character then a character might be quite long - I guess
> about ten codepoints

For actual real-life examples, much less than that.

For Indic grapheme clusters (which use mostly base characters,
not combining characters), the number can indeed get to around 10.

In theory, there is no limitation as to how many combining characters
can follow a base character.

> which can be themselves up to six bytes.

No, just up to four.

Regards,   Martin.

#-#-# Martin J. Du"rst, Assoc. Professor, Aoyama Gakuin University
#-#-# http://www.sw.it.aoyama.ac.jp   mailto:duerst@it.aoyama.ac.jp

=end

#20 - 02/24/2009 10:21 PM - hramrach (Michal Suchanek)

=begin
2009/2/24 Michael Selig michael.selig@fs.com.au:

> On Mon, 23 Feb 2009 21:35:30 +1100, Michal Suchanek hramrach@centrum.cz
> wrote:
>
> > Theoretically if separate (combining) character accents are considered
> > part of the character then a character might be quite long - I guess
> > about ten codepoints which can be themselves up to six bytes. However,
> > the number of accents one can put together should be limited to
> > meaningful combinations so this should still be secure - as long as
> > the code which determines what is a valid character does not have

> > > bugs. This might be tricky in some cases, though.
> >
> > BTW the same goes for "reading N bytes up to a character boundary"
> > unless you are willing to accept that you might read nothing even if
> > data is available because the character did not fit into N bytes, even
> > for a large N.
>
> The current behaviour of IO#gets "limit" parameter is "read N bytes but
> round *UP* to the next character boundary". Therefore you may get more bytes
> returned than requested. As long as "limit" is 1 or more, you should always
> read something unless the file is at EOF, right?
>
> However I still do not understand why reading N characters (instead of the
> current "limit" implementation) might be described as being "insecure". Can
> someone please explain it? It is clear that the string returned may be
> larger than N bytes long, but why is that "insecure"?

The current gets is not any more secure since it rounds up. If you
found a bug in an encoding that could give you an infinite character
either version would break.

Thanks

Michal

=end

#21 - 02/25/2009 10:54 PM - hramrach (Michal Suchanek)

=begin
2009/2/24 Michael Selig michael.selig@fs.com.au:

> Hi Michal,
>
> On Wed, 25 Feb 2009 00:20:52 +1100, Michal Suchanek hramrach@centrum.cz
> wrote:
>
> > > BTW the same goes for "reading N bytes up to a character boundary"
> > > unless you are willing to accept that you might read nothing even if
> > > data is available because the character did not fit into N bytes, even
> > > for a large N.
> >
> > The current behaviour of IO#gets "limit" parameter is "read N bytes but
> > round *UP* to the next character boundary". Therefore you may get more
> > bytes
> > returned than requested. As long as "limit" is 1 or more, you should
> > always
> > read something unless the file is at EOF, right?
> >
> > However I still do not understand why reading N characters (instead of
> > the
> > current "limit" implementation) might be described as being "insecure".
> > Can
> > someone please explain it? It is clear that the string returned may be
> > larger than N bytes long, but why is that "insecure"?
>
> The current gets is not any more secure since it rounds up. If you
> found a bug in an encoding that could give you an infinite character
> either version would break.

Actually rounding *up* to the character boundary *is* more secure. You
pointed it out yourself! If it rounded down, gets could return an empty
string, and code like:

```
while s = f.gets(1) .....
```

would then go into an infinite loop.

Then either way is insecure because you can get an infinite loop with
zero read (unless zero read returned nil or threw an exception) and
potentially infinite memory requirement with a broken encoding and
rounding up.

Thanks

Michal

=end

#22 - 04/04/2010 01:08 AM - znz (Kazuhiro NISHIYAMA)

- Category set to core

- Target version set to 2.0.0

=begin

=end

#23 - 09/14/2010 04:47 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

=begin

=end

#24 - 02/13/2012 09:05 PM - mame (Yusuke Endoh)

Are there any volunteers to summarize the discussion?
I cannot understand the whole discussion about this ticket.
Redmine failed to capture some mails.

--
Yusuke Endoh mame@tsg.ne.jp

#25 - 10/27/2012 04:41 AM - ko1 (Koichi Sasada)

- Target version changed from 2.0.0 to 2.6

I changed the target "next minor" this ticket because no response here.

#26 - 10/19/2017 01:03 PM - mame (Yusuke Endoh)

- Status changed from Assigned to Rejected

I'm rejecting this issue since it has been stalled for five years. If anyone really needs it, it would be good to re-organize the discussion all over again.