Ruby trunk - Bug #15210

UTF-8 BOM should be removed from String in internal representation

10/06/2018 06:47 PM - foonlyboy (Eike Dierks)

Status:	Open		
Priority:	Normal		
Assignee:	docs		
Target version:			
ruby -v:		**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

Hi everyone working on the ruby trunk,

I encountered a problem with a BOM (Byte Order Mark) at the front of UTF-8 string data.

We import some CSV from paypal.
They now include a BOM in front of their UTF-8 encoded CSV data.
This BOM is making some troubles.

I believe this to be a bug in how byte data is converted to the ruby internal String representation.

There is a workaround, but this needs to be documented:

`IO.read(mode:'r:BOM|UTF-8')`

But I'm asking for to improve the UTF-BOM handling:

- The BOM is only used for transfer encoding at the byte stream level.
- The BOM MUST NOT be part of the String in internal representation.

BTW: stdlib::CSV chokes on the BOM

I'd like to add some code for a workaround:

```
class String

    # delete UTF Byte Order Mark from string
    # returns self (even if no bom was found, contrary to delete_prefix!)
    # NOTE: use with care: better remove the bom when reading the file
    def delete_bom!
        raise 'encoding is not UTF-8' unless self.encoding == Encoding::UTF_8
        delete_prefix!("\xEF\xBB\xBF")
        return self
    end


    # returns a copy of string with UTF Byte Order Mark deleted from string
    def delete_bom
        dup.delete_bom!
    end

end
```

~eike

History

#1 - 10/06/2018 07:51 PM - shevegen (Robert A. Heiler)

> BTW: stdlib::CSV chokes on the BOM

I can't say how common this is or whether there is a bug; but in the event
that there may be, and the use case or situation involving the bug or faulty
behaviour affecting other ruby hackers, I would agree in this event that CSV
should probably be able to handle BOM-specific entries as well, in one way
or another (be it automatic or via another API).

I also agree that it could perhaps be mentioned somewhere, be it in the
csv documentation or elsewhere.

To the workaround: I assume you meant this only for a solution if others face
a similar problem, rather than a permanent addition to class String, yes?
(I ask this because adding a specific method to class String permanently in
ruby may be much harder to do and get approved, whereas an extension to ruby's
CSV is most likely easier and possible.)

#2 - 10/07/2018 12:53 AM - nobu (Nobuyoshi Nakada)

- Assignee set to docs

- Description updated

foonlyboy (Eike Dierks) wrote:

> I believe this to be a bug in how byte data is converted to the ruby internal String representation.

Yes, a BOM should be removed at the conversion, the reading from a data stream.

> There is a workaround, but this needs to be documented:
>
> ```
> IO.read(mode:'r:BOM|UTF-8')
> ```

It is documented at IO.new, and you can use it at CSV.open too.

rdoc of CSV.open:

> You must pass a filename and may optionally add a mode for Ruby's open().

rdoc of Kernel.open:

> See the documentation of IO.new for full documentation of the mode string directives.

rdoc of IO.new:

> If "BOM|UTF-8", "BOM|UTF-16LE" or "BOM|UTF16-BE" are used, Ruby checks for
> a Unicode BOM in the input document to help determine the encoding. For
> UTF-16 encodings the file open mode must be binary. When present, the BOM
> is stripped and the external encoding from the BOM is used. When the BOM
> is missing the given Unicode encoding is used as ext_enc. (The BOM-set
> encoding option is case insensitive, so "bom|utf-8" is also valid.)

Documents improvement patches are welcome.

> But I'm asking for to improve the UTF-BOM handling:
>
> - The BOM is only used for transfer encoding at the byte stream level.

This is half true.

https://en.wikipedia.org/wiki/Byte_order_mark#Usage

> If the BOM character appears in the middle of a data stream, Unicode says it should be interpreted as a "zero-width non-breaking space"

The character at other place is not called as "BOM".

> - The BOM MUST NOT be part of the String in internal representation.

Yes, it should be removed at the reading, that is the only chance to remove a BOM properly.

#3 - 10/12/2018 06:44 PM - foonlyboy (Eike Dierks)

I looked into it a bit more closely into it:

io.c does this in

```
static int
io_strip_bom(VALUE io)
```

which is called by:

```
static void
io_set_encoding_by_bom(VALUE io)
```

> It is documented at IO.new, and you can use it at CSV.open too.
> Yes, I was aware of this.

I also agree the the conversion has to take place at opening the file.

But with rails I get a ActionDispatch::Http::UploadedFile
(which returns an ASCII-8BIT byte stream)

And I could find no way to apply the io_strip_bom() to it,
not even by going through StringIO.
(but then Ruby is not about applying tricks anyway)

It sounds to me that nobu also agrees, that the BOM should always be removed.

> If the BOM character appears in the middle of a data stream, Unicode says it should be interpreted as a "zero-width non-breaking space"

I don't care so much about this for now.
(while I can imagine this to happen when concatenating files ...)

But let's fix the more simple problems first.

I think the BOM is used for two reasons in byte streams:

- a magic number for UTF encoded data (which might even apply to UTF-8)
- a magic number to distinguish different UTF byte orderings when using UTF-16, UTF-32, UTF-36?

But in the ruby world, we have **String**
We should remove all artefacts from any external encoding.

Impact:

I believe this might need a lot of changes throughout more than just one place in the code,
but I believe this should be fully upward compatible with *most* customers code.

This should still agree with the ruby spec,
because nowhere was it ever declared that String keeps the BOM.

Please excuse my lengthy writings,
but I thought these encoding problems were a thing from the past.

We might also look at the other languages around.
Makes for a good rosetta code ...

~eike