Ruby trunk - Feature #2567

Net::HTTP does not handle encoding correctly

01/07/2010 06:20 AM - slide_rule (Ryan Sims)

Status:	Assigned	
Priority:	Normal	
Assignee:	naruse (Yui NARUSE)	
Target version:		

Description

=begin
A string returned by an HTTP get does not have its encoding set appropriately with the charset field, nor does the content_type report the charset. Example code demonstrating incorrect behavior is below.

```
#!/usr/bin/ruby -w
# encoding: UTF-8

require 'net/http'

uri = URI.parse('http://www.hearya.com/feed/')
result = Net::HTTP.start(uri.host, uri.port) {|http|
http.get(uri.request_uri)
}

p result['content-type']     # "text/xml; charset=UTF-8" <- correct
p result.content_type        # "text/xml" <- incorrect; truncates the charset field
puts result.body.encoding    # ASCII-8BIT <- incorrect encoding, should be UTF-8
```
=end

Related issues:

Has duplicate Ruby trunk - Bug #15517: Net::HTTP not recognizing valid UTF-8	**Open**

History

#1 - 01/07/2010 01:14 PM - naruse (Yui NARUSE)

=begin
You can get Content-Type's parameter by result.type_params

I'm nervous about setting encoding to result body of net/http
because charset of Content-Type doesn't show true encoding of the content.
http://wiki.whatwg.org/wiki/Web_Encodings
=end

#2 - 01/10/2010 03:40 PM - naruse (Yui NARUSE)

- Category set to lib

- Priority changed from Normal to 3

=begin

=end

#3 - 04/19/2010 10:12 PM - mame (Yusuke Endoh)

- Assignee set to naruse (Yui NARUSE)

- Target version set to 2.0.0

=begin
Hi,

I'm neutral for guessing encodings by Content-Type.
I could be optimistic because I have never seen the
abyss of encodings :-)

Anyway, currently there is no feature that guesses

encodings from Content-Type. It is not a bug.
This ticket can be considered as feature request.
I moved the ticket to Feature tracker.

--
Yusuke Endoh mame@tsg.ne.jp
=end

#4 - 04/20/2010 02:36 AM - tenderlovemaking (Aaron Patterson)

=begin
Setting encoding based on 'Content-Type' header scares me. It is common that the webserver will report one encoding, and the HTML will contain a meta tag indicating a *completely different* encoding. The encoding indicated in the http header could conflict with the encoding in the HTML document. It seems safer to let clients of 'net/http' figure out which encoding is correct.
=end

#5 - 05/02/2010 12:32 AM - cabo (Carsten Bormann)

=begin
The previous comments appear to be confused. If the web server indicates a charset in an HTTP Content-Type header, this takes precedence over everything that may be in the body, so it is always correct to set the Ruby "encoding" from that. There is no "guessing" of the charset in that case.

Only if no charset is given in HTTP, would a browser resort to looking into the content for clues (META/HTTP-EQUIV in HTML4, META/CHARSET in HTML5).
If there are none, then a locale-specific default is likely to apply (contrary to the obsolete specification in RFC 2616, which mandates ISO-8859-1 as the standard default for this case; this is being removed in httpbis).

I'm neutral on whether Net::HTTP should attempt to deliver bodies in the correct Ruby "encoding" or always say binary = "ASCII-8BIT". The user of the body may want to apply content-type sniffing regardless of what the body is declared to be (e.g., to detect videos, zip/rar files, etc.). I just want to point out that the above comments give an incorrect reason not to operate based on a charset attribute of a Content-Type HTTP header.

A good, stable, fully vetted tutorial on character encoding on the Web is in

http://www.w3.org/International/tutorials/tutorial-char-enc/

The sniffing issue is the subject of an ongoing IETF internet-draft:

http://tools.ietf.org/html/draft-abarth-mime-sniff
=end

#6 - 05/04/2010 05:52 PM - naruse (Yui NARUSE)

=begin
The problem is stated in HTML5: misinterpreted for compatibility.
http://www.w3.org/TR/html5/syntax.html#character-encodings-0
=end

#7 - 09/14/2010 04:08 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

=begin

=end

#8 - 07/26/2011 10:56 AM - jrochkind (jonathan rochkind)

how else is a developer/client going to figure out the encoding EXCEPT from the HTTP server response content-type? You pretty much need to set the encoding in ruby 1.9 for the response to be useable. If the Content-Type isn't trustworthy enough for Net::HTTP... what else is available for the developer? In the vast majority of cases, the developer will have to map from content-type.

If Net::HTTP doesn't do it, then every developer has to figure out how to do it on their own and write extra code to do it. Setting the encoding from the HTTP response is going to be by far the most common use pattern. I think Net::HTTP should do it.

If there are certain edge cases where a knowledgeable developer knows s/he wants something different, then perhaps a "set_encoding?" attribute can be provided set to true by default, but settable to false. Alternately, the developer can always call force_encoding themselves as well.

The knowledgeable developer perhaps needs ways to disable this behavior, but the beginning developer needs reasonable behavior as a default, and right now they don't get it -- character encoding is one of the most confusing issues in ruby 1.9 (esp to debug), and the more standard libraries set it appropriately when appropriate information is available, the less of a barrier this is to getting started with ruby 1.9. When you're reading from an arbitrary data stream without encoding metadata (like the file system), it's one thing, the standard libraries can't possibly know what to do. But when you're reading from a stream with encoding metadata (HTTP headers), using a library specifically designed for that kind of stream (stdlib Net::HTTP), it's unforgiveable that ruby makes each (newbie!) developer reinvent the wheel again and again.

#9 - 07/26/2011 04:02 PM - drbrain (Eric Hodel)

The problem is not so much forcing the user to figure out how to get correct encoding (charset) but making sure the encoding returned is accurate. If we can add this feature to Net::HTTP in a way that works for most cases that's great.

Unfortunately websites outside of the US seem to have big problems with guessing the encoding correctly and require an attempt at parsing the document first. Most bugs in mechanize about setting the encoding correctly came from people parsing non-English and non-Latin websites (so UTF-8 or ISO-8859-1 won't work).

If we can do this without needing to parse the document that's great, but I think that is very difficult to do. Having a broken or inaccurate way of choosing the encoding will be worse than having no way.

#10 - 11/19/2011 12:40 PM - mksm (Ricardo Amorim)

Yui NARUSE wrote:

> The problem is stated in HTML5: misinterpreted for compatibility.
> http://www.w3.org/TR/html5/syntax.html#character-encodings-0

That link is stale as the specification is constantly being improved. I suggest http://www.w3.org/TR/html5/parsing.html#determining-the-character-encoding for discussing. The first few points should be implemented in Net::HTTP, IMHO.

#11 - 11/22/2011 01:46 PM - drbrain (Eric Hodel)

=begin
What should the user expect when the response headers are wrong? For example, the response Content-Type claims ISO-8859-1 but the content was UTF-8? (Yes, this really happens)

If Net::HTTP forces the encoding to ISO-8859-1 you will have undetectably garbled text:

```
$ ruby -e 's = "π"; s.force_encoding Encoding::ISO_8859_1; puts s.valid_encoding?, s.encode(Encoding::UTF_8)'
true
ï
```

I think leaving the response as binary encoding and allowing the user to apply the proper heuristics to determine the encoding for their is the best way.

If you wish to read HTML documents, perhaps mechanize is a better choice as it implements a heuristic similar to the one in HTML5 to find the encoding of the document despite potential lies from the server or document header.

=end

#12 - 11/22/2011 07:03 PM - regularfry (Alex Young)

Surely setting the encoding to whatever the content-type header declares doesn't stop mechanize from performing that heuristic? Setting it to binary (incorrectly, in my view) forces me to fix it manually even when I know everything's lined up properly. Worse, in order to do it, I have to string match on the content-type header itself, when Net::HTTP has already done that work and has the information available.

If I'm *not* reading HTML documents (which is *far* from uncommon, certainly in my case), the idea of having to apply a heuristic on top of Net::HTTP just doesn't make sense: the information is there in the headers, and Net::HTTP is best placed to interpret it. It's not the transport layer's job to make assumptions about the content it's transporting.

#13 - 11/22/2011 08:20 PM - Eregon (Benoit Daloze)

I agree with Alex Young, the encoding should be set from the header if available.

I believe a sensible default is way better than requiring the user to do the obvious. One can always use force_encoding if he knows the header is wrong, in which case the error if from the page, not Net:::HTTP.

#14 - 11/23/2011 02:54 AM - drbrain (Eric Hodel)

So giving the user undetectably garbled text is acceptable to both of you? I wish to clarify.

If the Content-Type header is used as you propose and the user sets the default_internal encoding what should happen? If the server lies and the response body is transcoded data may be lost or an exception may be raised. Should this exception be rescued by Net::HTTP? What should the result encoding be if it is?

#15 - 11/23/2011 09:00 PM - regularfry (Alex Young)

Eric Hodel wrote:

> So giving the user undetectably garbled text is acceptable to both of you? I wish to clarify.

Yes. If the user is getting garbled text, *they'll see it* and can fix it. It's only undetectable at the data level - once it gets rendered, it's obvious.

> If the Content-Type header is used as you propose and the user sets the default_internal encoding what should happen?

Use the Content-Type header. Only use default_internal if the Content-Type doesn't have specify a charset.

> If the server lies and the response body is transcoded data may be lost or an exception may be raised. Should this exception be rescued by Net::HTTP?

No, it should be left uncaught, or re-raised, to tell the user that there's breakage in the encoding pipeline. As long as there's a mechanism the user can use to *opt* to ignore the server's charset to help diagnosing this sort of breakage, I don't see that this is problematic.

#16 - 11/24/2011 01:57 AM - Eregon (Benoit Daloze)

Eric Hodel wrote:

> So giving the user undetectably garbled text is acceptable to both of you? I wish to clarify.

Yes, as it should be garbled only when the response has a wrong Content-Type, in which case the user needs to check if it is the right encoding anyway.

(And AFAIK, Firefox always reported garbled text if I set the meta tag to the right encoding and the Content-Type header to the wrong encoding in my tries.)

> If the Content-Type header is used as you propose and the user sets the default_internal encoding what should happen?

I think leaving it as BINARY (as now) is fine in this case. Assuming default_internal is the right encoding does not seem to be a good heuristic.

> If the server lies and the response body is transcoded data may be lost or an exception may be raised. Should this exception be rescued by Net::HTTP? What should the result encoding be if it is?

I think Net::HTTP should not transcode (#encode) the response, just set the right encoding if the information is available.

#17 - 11/24/2011 02:15 AM - naruse (Yui NARUSE)

I don't decide whether merge this or not yet, an experimental patch is following:

```
diff --git a/lib/net/http.rb b/lib/net/http.rb
index 1c594e0..0abcaa5 100644
--- a/lib/net/http.rb
+++ b/lib/net/http.rb
@@ -2723,6 +2723,8 @@ module Net   #:nodoc:
       end
       @read = true

+      enc = detect_encoding(@body)
+      @body.force_encoding(enc) if enc
       @body
     end

@@ -2807,6 +2809,167 @@ module Net   #:nodoc:
       end
     end

+    private
+    # :nodoc:
+    def detect_encoding(str, encoding=nil)
+      if encoding
+      elsif encoding = type_params['charset']
+      elsif encoding = check_bom(str)
+      else
+        case main_type.downcase
+        when %r{text/x(?:ht)?ml|application/(?:[^+]+\+)?xml}
+          /\A<xml[ \t\r\n]+
+            version[ \t\r\n]*=[ \t\r\n]*(?:"[0-9.]+"|'[0-9.]*')[ \t\r\n]+
+            encoding[ \t\r\n]*=[ \t\r\n]*
+             (?:"([A-Za-z][\-A-Za-z0-9._]*)"|'([A-Za-z][\-A-Za-z0-9._]*)')/x =~ str
+          encoding = $1 || $2 || Encoding::UTF_8
+        when %r{text/html.*}
+          sniff_encoding(str, encoding=nil)
+        end
+      end
+      return encoding
+    end
+
+    # :nodoc:
+    def sniff_encoding(str, encoding=nil)
+      # the encoding sniffing algorithm
+      # http://www.w3.org/TR/html5/parsing.html#determining-the-character-encoding
+      return enc if enc = scanning_meta(str)
+      # 6. last visited page or something
+      # 7. frequency
+      if str.ascii_only?
+        return Encoding::US_ASCII
+      else
+        utf8str = str.dup.force_encoding(Encoding::UTF_8)
+        return utf8str if utf8str.valid_encoding?
+      end
+      # 8. implementation-defined or user-specified
+    end
+
+    # :nodoc:
+    def check_bom(str)
+      case str.byteslice(0, 2)
+      when "\xFE\xFF"
+        return Encoding::UTF_16BE
+      when "\xFF\xFE"
+        return Encoding::UTF_16LE
+      end
+      if "\xEF\xBB\xBF" == str.byteslice(0, 3)
+        return Encoding::UTF_8
+      end
+      nil
+    end
+
+    # :nodoc:
+    def scanning_meta(str)
+      require 'strscan'
+      ss = StringScanner.new(str)
+      while true
+        if ss.skip(/<!--.*?-->/)
+        elsif ss.skip(/meta[\t\n\f\r ]*/)
+          attrs = {} # attribute_list
+          got_pragma = false
+          need_pragma = nil
+          charset = nil
+
+          # step: Attributes
+          while attr = get_attribute(ss)
+            name, value = *attr
+            next if attrs[name]
+            attrs[name] = true
+            case name
+            when 'http-equev'
+              got_pragma = true if value == 'content-type'
+            when 'content'
+              encoding = extracting_encodings_from_meta_elements(value)
+              unless charset
+                charset = encoding
+              end
+              need_pragma = true
+            when 'charset'
+              need_pragma = false
+              charset = value
+            end
+          end
+
+          # step: Processing
+          next if need_pragma.nil?
+          next if need_pragma && !got_pragma
+          charset = Encoding.find(charset) rescue nil
+          next unless charset
+          charset = Encoding::UTF_8 if charset == Encoding::UTF_16
+          return charset # tentative
+        elsif ss.skip(/<\/?[A-Za-z][^\t\n\f\r ]*/)
+          1 while get_attribute(ss)
+        elsif ss.skip(/<[!\/?][^>]*>/)
+        elsif ss.getch
+        end
+      end
+      nil
+    end
+
+    def get_attribute(ss)
+      ss.scan(/[\t\n\f\r \/]*/)
+      if ss.peek(1) == '>'
+        ss.getch
+        return nil
+      end
+      name = ss.scan(/[^=\t\n\f\r \/>]*/)
+      name.downcase!
+      raise if name.empty?
+      ss.skip(/[\t\n\f\r ]*/)
+      if ss.getch != '='
+        value = ''
+        return [name, value]
+      end
+      ss.skip(/[\t\n\f\r ]*/)
+      case ss.peek(1)
+      when '"'
+        ss.getch
+        value = ss.scan(/[^"]+/)
+        value.downcase!
+        ss.getch
+      when "'"
+        ss.getch
+        value = ss.scan(/[^']+/)
+        value.downcase!
+        ss.getch
+      when '>'
+        value = ''
+      else
+        value = ss.scan(/[^\t\n\f\r >]+/)
+        value.downcase!
+      end
+      [name, value]
+    end
+
+    def extracting_encodings_from_meta_elements(value)
+      # http://dev.w3.org/html5/spec/fetching-resources.html#algorithm-for-extracting-an-encoding-from-a-meta
-element
+      if /charset[\t\n\f\r ]*=(?:"([^"]*)"|'([^']*)'|["']|\z|([^\t\n\f\r ;]+))/i =~ value
+        return $1 || $2 || $3
+      end
+      return nil
+    end
+
+    # http://dev.w3.org/html5/spec/parsing.html#table-encoding-overrides
+    TABLE_ENCODING_OVERRIDES = {
+      'EUC-KR'         => Encoding::CP949,
+      'EUC-JP'         => Encoding::CP51932,
+      'GB2312'         => Encoding::GBK,
+      'GB_2312-80'     => Encoding::GBK,
+      'ISO-8859-1'     => Encoding::Windows_1252,
+      'ISO-8859-9'     => Encoding::Windows_1254,
+      'ISO-8859-11'    => Encoding::Windows_874,
+      'KS_C_5601-1987' => Encoding::CP949,
+      'SHIFT_JIS'      => Encoding::Windows_31J,
+      'TIS-620'        => Encoding::Windows_874,
+      'US-ASCII'       => Encoding::Windows_1252,
+    }
+
+    # :nodoc:
+    def override_encoding(enc)
+      TABLE_ENCODING_OVERRIDES[enc.strip.upcase] || enc
+    end
   end
```

#18 - 11/24/2011 07:53 AM - nahi (Hiroshi Nakamura)

On Thu, Nov 24, 2011 at 02:15, Yui NARUSE naruse@airemix.jp wrote:

> I don't decide whether merge this or not yet, an experimental patch is following:

Good challenge!

There's an implementation in open-uri.rb. You'd better check that for merge.

Just saying above. I have no idea about adding this feature to
Net::HTTP. Net::HTTP should find a maintainer first...

#19 - 11/24/2011 08:23 AM - naruse (Yui NARUSE)

(2011/11/24 7:38), Hiroshi Nakamura wrote:

> There's an implementation in open-uri.rb. You'd better check that for merge.

open-uri's is pure HTTP one, mine is HTML5's.

> Just saying above. I have no idea about adding this feature to
> Net::HTTP. Net::HTTP should find a maintainer first...

I don't think net/http need a maintainer unless someone want to do big change.
If someone want to do, they should be.

--
NARUSE, Yui naruse@airemix.jp

#20 - 11/24/2011 11:12 AM - mksm (Ricardo Amorim)

Benoit Daloze wrote:

> I think Net::HTTP should not transcode (#encode) the response, just set the right encoding if the information is available.

I agree with the above and with Alex. Net::HTTP should only set the encoding (#force_encoding) by checking Content-type or the HTML5 heuristics (that Naruse seemed to have implemented). If the response body becomes garbled then the user should clearly see it and decide on what to do.

#21 - 11/24/2011 11:25 AM - mksm (Ricardo Amorim)

Also, response header values encoding are set to ASCII-8BIT. According to this: http://tools.ietf.org/html/rfc2616#section-2.2, it seems the default is ISO-8859-1. Having the values in ASCII-8BIT can cause issues when parsing the "Location" header with URI.

#22 - 11/24/2011 11:56 AM - naruse (Yui NARUSE)

Ricardo Amorim wrote:

> Also, response header values encoding are set to ASCII-8BIT. According to this: http://tools.ietf.org/html/rfc2616#section-2.2, it seems the default is ISO-8859-1. Having the values in ASCII-8BIT can cause issues when parsing the "Location" header with URI.

It shouldn't effect because URI doesn't include non ASCII character.
If you are talking about an existing implementation which sends Location header with non ASCII characters,
such talk should be on real research.

#23 - 11/24/2011 01:01 PM - mksm (Ricardo Amorim)

Yui NARUSE wrote:

> It shouldn't effect because URI doesn't include non ASCII character.
> If you are talking about an existing implementation which sends Location header with non ASCII characters,
> such talk should be on real research.

I've seen a few ASP applications that do that. They redirect to a generic error page with an error message as an argument. e.g. below:
Location: error_page.asp?msg="Página com erro".

Well doing some research I've found:
http://tools.ietf.org/html/draft-ietf-httpbis-p1-messaging-17#section-3.2.1

"Historically, HTTP has allowed field content with text in the ISO-
8859-1 [ISO-8859-1] character encoding and supported other character

sets only through use of [RFC2047] encoding. In practice, most HTTP
header field values use only a subset of the US-ASCII character
encoding [USASCII]. Newly defined header fields SHOULD limit their
field values to US-ASCII octets. Recipients SHOULD treat other (obs-
text) octets in field content as opaque data."

It's not entirely clear if non US-ASCII chars are allowed in field contents.

#24 - 11/24/2011 07:53 PM - naruse (Yui NARUSE)

2011/11/24 Ricardo Amorim mksm@iddqd.su:

> Issue #2567 has been updated by Ricardo Amorim.
>
> Yui NARUSE wrote:
>
>> It shouldn't effect because URI doesn't include non ASCII character.
>> If you are talking about an existing implementation which sends Location header with non ASCII characters,
>> such talk should be on real research.
>
> I've seen a few ASP applications that do that. They redirect to a generic error page with an error message as an argument. e.g. below:
> Location: error_page.asp?msg="Página com erro".

Is such a string always ISO-8859-1 other than non US/West Europe?

> Well doing some research I've found:
> http://tools.ietf.org/html/draft-ietf-httpbis-p1-messaging-17#section-3.2.1
>
> "Historically, HTTP has allowed field content with text in the ISO-
> 8859-1 [ISO-8859-1] character encoding and supported other character
> sets only through use of [RFC2047] encoding. In practice, most HTTP
> header field values use only a subset of the US-ASCII character
> encoding [USASCII]. Newly defined header fields SHOULD limit their
> field values to US-ASCII octets. Recipients SHOULD treat other (obs-
> text) octets in field content as opaque data."
>
> It's not entirely clear if non US-ASCII chars are allowed in field contents.

The sentence seems to talk about fields which allow non ASCII but usually
use only ASCII. So Location is also on this context.
And the encoding for "opaque data" should be ASCII-8BIT.

--
NARUSE, Yui naruse@airemix.jp

#25 - 11/25/2011 12:54 AM - mksm (Ricardo Amorim)

Yui NARUSE wrote:

> Is such a string always ISO-8859-1 other than non US/West Europe?

Yes, ISO-8859-1 always fits. I'm mainly accessing Brazilian servers so that explains.

> The sentence seems to talk about fields which allow non ASCII but usually
> use only ASCII. So Location is also on this context.
> And the encoding for "opaque data" should be ASCII-8BIT.

With some more research, I got to this: http://tools.ietf.org/html/rfc2396, topic 2.1.

So, I think you are correct. When the URI is decoded into an octet stream the correct encoding is unknown and has no default. Looks like ASCII-8BIT is the best choice but certainly people WILL have issues when redirecting to non US-ASCII URL's.

#26 - 11/25/2011 12:06 PM - naruse (Yui NARUSE)

Ricardo Amorim wrote:

> Yui NARUSE wrote:
>
>> Is such a string always ISO-8859-1 other than non US/West Europe?

> Yes, ISO-8859-1 always fits. I'm mainly accessing Brazilian servers so that explains.

As I understand, Brazilian uses Portuguese and it is in ISO-8859-1.

Anyway, I found a description about deciding encoding on http-bis.
http://tools.ietf.org/html/draft-ietf-httpbis-p3-payload-17#section-4.2

In practice, resource owners do not always properly configure their
origin server to provide the correct Content-Type for a given
representation, with the result that some clients will examine a
response body's content and override the specified type. Clients
that do so risk drawing incorrect conclusions, which might expose
additional security risks (e.g., "privilege escalation").
Furthermore, it is impossible to determine the sender's intent by
examining the data format: many data formats match multiple media
types that differ only in processing semantics. Implementers are
encouraged to provide a means of disabling such "content sniffing"
when it is used.

So to discourage developers' net/http should set an encoding until it is practical.

#27 - 05/17/2012 08:38 AM - jrochkind (jonathan rochkind)

It seems like encoding on headers is a different question than encoding on bodies.

Perhaps encoding on headers should be left ascii-8bit -- I don't understand if the spec even says the charset in the header is supposed to apply to other headers.

But it is clear to me that encoding on body should be set per headers, when possible.

Most langauges are not so explicit about encoding as ruby 1.9. In most languages you can get away with ignoring encoding, and at worst get garbled text -- in ruby 1.9 you'll get exceptions raised.

'Implementers are encouraged to provide a means of disabling such "content sniffing"   when it is used.'

Fortunately, there is a clear way to do that -- we're not talking about net::http doing any transcoding, only about it setting the encoding value. You want to 'disable' that? Just

`response.body.force_encoding("ASCII-8BIT")`

to throw out whatever encoding it determined from the headers.

Whatever problems would be caused by http servers sending bad content-type header and net::http believing it -- wouldn't those same problems also be caused by leaving the encoding ASCII-8BIT? If an individual client wants to use heuristics to guess encoding, there's nothing stopping them -- just force_encoding("ASCII-8BIT") and then use whatever heuristics you like and force_encoding as a result at the end.

But by default, net::http should assume that the spec is being followed and the content-type header is correct.

#28 - 05/17/2012 08:41 AM - jrochkind (jonathan rochkind)

It actually occurs to me that I mis-read the passage quoted by naruse.

That passage is discouraging heuristical guessing of charset, despite the fact that content-type is often wrong. That's what's being discouraged, and what it's saying there should be an opt-out of.

#29 - 11/20/2012 10:43 PM - mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 2.6

#30 - 11/20/2012 10:44 PM - mame (Yusuke Endoh)

- Target version changed from 2.6 to 2.0.0

#31 - 02/17/2013 11:42 PM - naruse (Yui NARUSE)

- Target version changed from 2.0.0 to 2.6

#32 - 03/11/2014 07:34 PM - meta (mathew murphy)

This still seems to be a problem in Ruby 2.1.1p76. Data ends up as ASCII-8BIT even if the server specifies that it's UTF-8 or ISO-8859-1, and I end up with errors when writing it out somewhere else as a result.

#33 - 07/22/2014 05:29 PM - hugo.corbucci (Hugo Corbucci)

I've just hit this problem again.
I've read all comments and it seems like I see 3 different opinions:
1) Content type is unrealiable so clients of Net::HTTP should force the encoding to whatever they want whenever they want to use the body of a response.
2) Content type is unrealiable but that's the webserver's fault so Net::HTTP should force the encoding of the body to whatever content type specifies if any or the default_encoding otherwise. Clients who are accessing an unrealiable webserver should force the encoding.
3) Content type is unrealiable so Net::HTTP should try to detect the encoding from the body and then force the body into whatever is found or default_encoding otherwise.

1) requires no work and is the currently implemented solution.
2) needs a patch which is a subset of the one posted by NARUSE.
3) needs a patch which is something close to NARUSE's suggestion (if not all of it).

Changing from 1) to 2) causes a breaking change for every user of Net::HTTP that doesn't currently force the encoding and relies on it being ASCII-8BIT.
Changing from 1) to 3) causes a breaking change in some cases (the ones where the detection algorithm is wrong) if the user of Net::HTTP doesn't currently force the encoding.

Seems to me that this means if a user is properly using the solution in 1), changing it to either 2 or 3 doesn't affect anything. If the user is not forcing the encoding, then there is already a potential problem waiting to happen.

I would honestly prefer Net::HTTP to rely on the data provided by the server both for the body meaning I would consider the header it sent along with the body to inform me of the correct data. If it doesn't, I need to act on this anyway. But if it behaves correctly, I don't have to do anything. Seems better than having to force me to do extra work even though all sides are behaving nicely.

What is stopping this feature from being implemented? A patch?

#34 - 09/22/2014 11:42 PM - marcel (Marcel Cary)

I'm also encountering this issue after upgrading from 1.8.7 to 2.0.0. The issue was difficult to troubleshoot because it didn't manifest until after Net::HTTP was done and gone from the stack, when trying to concatenate the response body with a UTF-8 string. It was also intermittent in that many responses did not trigger the error.

Although I confess that I've never had to deal with a server that misidentifies the charset of the response, I lean toward Hugo Corbucci's assessment: Net::HTTP should use the content-type header and possibly the body. But if that's too risky, how about just adding a hook, something like Net::HTTP.charset_guesser, which defaults to setting ASCII-8BIT, but can easily be set to use the http header or scan the body instead? That way the default behavior doesn't change, but we get an easy mechanism for setting Corbucci's #2 or #3 application wide on an opt-in basis, for example via loading a gem.

If a patch is what's stopping this feature from being implemented, I'm happy to provide one.

#35 - 03/08/2016 02:25 PM - fornellas (Fabio Pugliese Ornellas)

Hello,

I'm gonna give my 50 cents:

```
class Net::HTTPResponse
  def read_body(dest = nil, &block)
    if @read
      raise IOError, "#{self.class}\#read_body called twice" if dest or block
      return @body
    end
    # Force encoding for streamed response bodies
    final_block = if block
      proc do |chunk|
        if type_params['charset']
          block.call(chunk.force_encoding(type_params['charset']))
        else
          block.call(chunk)
        end
      end
    end
    to = procdest(dest, final_block)
    stream_check
    if @body_exist
      read_body_0 to
      @body = to
    else
      @body = nil
    end
    @read = true
    # Force encoding for String @body
    if type_params['charset'] && @body.respond_to?(:force_encoding)
      @body.force_encoding(response.type_params['charset'])
    end
    @body
  end
end
```

These changes:

- Makes Net::HTTP respect https://tools.ietf.org/html/rfc7231#section-3.1.1.2
- It woks for both cases: Net::HTTPResponse.body and Net::HTTPResponse.read_body.
- If there is there is a server misconfiguration, and content-type charset is different from response body, it will postpone encoding exceptions to body processing outside Net::HTTP code, thus making it clearer to the user.
- Users are still allowed to force_encoding to bypass any server misconfiguration.

I understand Ruby libraries must obey RFC's by default, and let users get real exceptions when something is not right. The way it is now, body strings come inconsistent: sometimes I get ASCII-8BIT, sometimes UTF-8, depending on how the code inside Net::HTTP runs, and the RFC is not obeyed.

I believe this change might create problems, with code that "works by coincidence", due to current behavior. For example, if the server is misconfigured, and set charset to iso8859-1, but response body is actually UTF-8, it will currently work, but with proposed patch, it will break. In such case however, it is a server issue, not client-side issue. It certainly is a risk, but not follow RFCs, is already bad as it is.

#36 - 10/22/2017 11:01 AM - chucke (Tiago Cardoso)

Bitten by this as well. I'd go the route proposed earlier:

1. By default, encode the body using the charset set in content-type header.
2. Provide an option to disable this, to keep old behaviour.

This way, people having to deal with "mis-behaving" servers have a way to "opt-in" on their legacy workaround, while everyone else gets the "least-surprise" resource.

#37 - 12/15/2017 08:22 AM - naruse (Yui NARUSE)

chucke (Tiago Cardoso) wrote:

> Bitten by this as well. I'd go the route proposed earlier:
>
> 1. By default, encode the body using the charset set in content-type header.

HTML's encoding is definition is bit different from usual encoding converters as described at WHATWG Encoding Standard.
https://encoding.spec.whatwg.org/

And charset parameter has many aliases which sometimes different from normal encoding aliases.
https://encoding.spec.whatwg.org/#names-and-labels

> 1. Provide an option to disable this, to keep old behaviour.

How the option is specified is problem.
The encoding may differ per content (URL / path).
Then it should be specified with get/post methods.
But there's already header and data hash arguments...

#38 - 01/13/2019 10:33 AM - naruse (Yui NARUSE)

- Has duplicate Bug #15517: Net::HTTP not recognizing valid UTF-8 added