Ruby master - Bug #12852

URI.parse can't handle non-ascii URIs

10/18/2016 08:10 PM - olivierlacan (Olivier Lacan)

Status:	Open		
Priority:	Normal		
Assignee:	akira (akira yamada)		
Target version:			
ruby -v:		**Backport:**	2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

Given a return URL path like: /search?utf8=\u{2713}&q=foo, URI.parse raises the following exception:

```
URI.parse "/search?utf8=\u{2713}&q=foo"
URI::InvalidURIError: URI must be ascii only "/search?utf8=\u{2713}&q=foo"
```

This \u{2713} character is commonly used by web frameworks like Rails to enforce UTF-8 in forms:
https://github.com/rails/rails/blob/92703a9ea5d8b96f30e0b706b801c9185ef14f0e/actionview/lib/action_view/helpers/form_tag_helper.rb#L823-L830

```
"\u{2713}"
=> "✓"
```

Is it unreasonable to expect non-ascii portion of URIs to be handled by URI.parse? The way to circumvent this issue is to call URI.encode on the URI string prior to passing it to URI.parse:

```
URI.parse URI.encode("/search?utf8=\u{2713}&q=foo")
=> #<URI::Generic /search?utf8=%E2%9C%93&q=foo>
```

By comparison, a library like Addressable parses this URI without issue.

```
require "addressable/uri"
=> #<Addressable::URI:0x3feffa84158c URI:/search?utf8=✓&q=foo>
```

This is how Addressable implements parsing:
https://github.com/sporkmonger/addressable/blob/a15b7045a09911bcc47b106200554809c879a5f6/lib/addressable/uri.rb#L75-L145

PS: Tried under MRI 2.3.1 and 2.4.0-preview1

History

#1 - 10/18/2016 09:18 PM - phluid61 (Matthew Kerwin)

As a point of order, there's no such thing as a "non-ASCII URI"*; that would be an IRI.

The rails snippet you linked is part of a HTML form. A web browser displaying and submitting that form would interpret the ✓ entity as U+2713 CHECK MARK, yes, but it would percent-encode it as %E2%9C%93 before using it in a HTTP request, because HTTP uses URIs, not IRIs. (The browser may present it as a single Unicode character in the awesomebar/omnibar/address bar, but that's a UI presentation element and not a true and accurate display of the URI.)

* see RFC 3296, Section 2

#2 - 10/18/2016 09:57 PM - olivierlacan (Olivier Lacan)

Matthew Kerwin wrote:

> The rails snippet you linked is part of a HTML form. A web browser displaying and submitting that form would interpret the ✓ entity as U+2713 CHECK MARK, yes, but it would percent-encode it as %E2%9C%93 before using it in a HTTP request, because HTTP uses URIs, not IRIs. (The browser may present it as a single Unicode character in the awesomebar/omnibar/address bar, but that's a UI presentation element and not a true and accurate display of the URI.)

It's common for OAuth authentication flows to store a destination URI to return to when the handshake process is completed. This URI can be stored without first being processed by a web sever that will encode it in the way Rails does for submitted forms since it's not meant to be processed — that is until it comes back to the origin server.

I opened this due to an issue I encountered in an OAuth provider handshake procedure. You could argue that I should be expected to URI.encode any URI set as a destination query parameter to prevent this issue from occurring, surely.

Do you not agree that URI.parse should accept unicode entities in URIs? It wasn't clear from your response.

I'm not aware of any IRI-compatible API in MRI that could allow me to directly parse URIs containing non-ASCII characters with Ruby, whether they match the strict definition of a URI or not.

#3 - 10/18/2016 11:27 PM - phluid61 (Matthew Kerwin)

Olivier Lacan wrote:

> It's common for OAuth authentication flows to store a destination URI to return to when the handshake process is completed. This URI can be stored without first being processed by a web sever that will encode it in the way Rails does for submitted forms since it's not meant to be processed — that is until it comes back to the origin server.

You keep using the word "URI" to refer to these data objects, and by specification a URI data object cannot contain non-ASCII characters (and even some ASCII characters are forbidden.) If we agree that "haha\nlol" is a String that cannot be parsed as a URI, we should agree the same for "http://example.org/\u{2713}".force_encoding('UTF-8')

> I opened this due to an issue I encountered in an OAuth provider handshake procedure. You could argue that I should be expected to URI.encode any URI set as a destination query parameter to prevent this issue from occurring, surely.

That's what I'm saying, but from the opposite direction. If you're storing a String, and want to ensure that it encodes a valid URI, you should URI.encode the parts before storing them in the String.

By analogy, if we replace "URI" with "JSON" in this discussion the same holds true: "{\"foo\":\"\n\"}" holds a String that looks a lot like JSON, but isn't valid [RFC 7159], and JSON.parse correctly raises an exception on it.

If a network peer is sending you a message that includes bytes that look like a URI but with UTF-8-encoded Unicode characters and *not* ASCII-compatible percent-encoded octets (i.e. it's sending an IRI), then one of two things is happening:

1. the protocol you're using is built on IRIs, not URIs, and you are responsible for any transformations to/from URIs (including URI.encode); or
2. the peer is in violation of a spec, and you should throw an error back at it. (In this case the specs are usually quiet clear on exactly what error to throw, too.)

> Do you not agree that URI.parse should accept unicode entities in URIs? It wasn't clear from your response.

I think URI.parse correctly raises an exception when it encounters characters that are forbidden by RFC3986.

> I'm not aware of any IRI-compatible API in MRI that could allow me to directly parse URIs containing non-ASCII characters with Ruby, whether they match the strict definition of a URI or not.

Your thinking here seems confused. If a String contains non-ASCII characters then it's not a URI. If it is a URI then it strictly matches the definition of a URI. If a String contains a valid IRI, then yeah, you're not going to get much help from Ruby; but IRIs are not commonly used in the real world anyway.

#4 - 12/12/2016 06:39 PM - naruse (Yui NARUSE)

Matthew Kerwin wrote:

> Your thinking here seems confused. If a String contains non-ASCII characters then it's not a URI. If it is a URI then it strictly matches the definition of a URI. If a String contains a valid IRI, then yeah, you're not going to get much help from Ruby; but IRIs are not commonly used in the real world anyway.

The concept sounds reasonable.
And I'm considering URL Standard's parsing logic is more suitable for Ruby's URI.parse.
https://url.spec.whatwg.org/
But the algorithm is still developing.