Ruby master - Feature #6752

Replacing ill-formed subsequencce

07/19/2012 11:42 AM - naruse (Yui NARUSE)

Status:	Closed
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	2.6

Description

=begin
== 概要
Stringになんらかの理由で不正なバイト列が含まれている時に、それを置換文字で置き換えたい。

== ユースケース
実際に確認されているユースケースは以下の通りです。

- twitterのtitle
- IRCのログ
- ニコニコ動画の API

- Webクローリング
  これらの不正なバイト列の生成過程は、おそらく、バイト単位で文字列を切り詰めた時に末尾が切れて、
  末尾がおかしい不正な文字列が作られます。（前二者）
  これをコンテナに入れたり結合することによって、途中にも混ざった文字列が作られます。（後二者）

- https://twitter.com/takahashim/status/18974040397
- https://twitter.com/n0kada/status/215674740705210368
- https://twitter.com/n0kada/status/215686490070585346
- https://twitter.com/hajimehoshi/status/215671146769682432
- http://po-ru.com/diary/fixing-invalid-utf-8-in-ruby-revisited/
- http://stackoverflow.com/questions/2982677/ruby-1-9-invalid-byte-sequence-in-utf-8

== 必要な引数: 置換文字
省略可能、String。
デフォルトは、Unicode系ならU+FFFD、それ以外では「?」。
デフォルトが空文字でない理由は、削除してしまうことで、従来は存在しなかったトークンを作れてしまい、
上位のレイヤーの脆弱性に繋がるからです。
http://unicode.org/reports/tr36/#UTF-8_Exploit

== API
--- str.encode(str.encoding, invalid: replace, [replace: "〓"])

- CSI的じゃなくて気持ち悪い
- iconv でできるのは glibc iconv か GNU libiconv に //IGNORE つけた時で他はできない
- 実装上のメリットは後述の通り、直感に反してあまりない(と思う)

== 別メソッド

- 新しいメソッドである
- fix/repair invalid/illegal bytes/sequence あたりの名前か

== 実装
=== 鬼車ベース
int ret = rb_enc_precise_mbclen(p, e, enc); して、
MBCLEN_INVALID_P(ret) が真な時、何バイト目が不正なのかわからないのが微妙。
ONIGENC_CONSTRUCT_MBCLEN_INVALID() がバイト数を取らないのが原因なので、

鬼車のエンコーディングモジュール全てに影響してしまうため、修正困難。
不正なバイトはほとんど存在しないと仮定して、効率を犠牲にすれば回避は可能。

=== transcodeベース
UCS正規化なglibc iconv, GNU libiconv, Perl Encodeなどと違って、
CSIなtranscodeでは、自分自身に変換する場合、
エンコーディングごとに「何もしない」変換モジュールを用意しないといけない。

とりあえず鬼車ベースのコンセプト実装とテストを添付しておきます。

```
diff --git a/string.c b/string.c
index d038835..4808f15 100644
--- a/string.c
+++ b/string.c
@@ -7426,6 +7426,199 @@ rb_str_ellipsize(VALUE str, long len)
return ret;
}

+/*
```

- * call-seq:
- *   str.fix_invalid -> new_str
- *
- * If the string is well-formed, it returns self.
- * If the string has invalid byte sequence, repair it with given replacement
- * character.
- */ +VALUE +rb_str_fix_invalid(VALUE str) +{
- int cr = ENC_CODERANGE(str);
- rb_encoding *enc;
- if (cr == ENC_CODERANGE_7BIT || cr == ENC_CODERANGE_VALID)
- return rb_str_dup(str); +
- enc = STR_ENC_GET(str);
- if (rb_enc_asciicompat(enc)) {
- const char *p = RSTRING_PTR(str);
- const char *e = RSTRING_END(str);
- const char *p1 = p;
- /* 10 should be enough for the usual use case,
- * fixing a wrongly chopped character at the end of the string
- */
- long room = 10;
- VALUE buf = rb_str_buf_new(RSTRING_LEN(str) + room);
- const char *rep;
- if (enc == rb_utf8_encoding())
- rep = "\xEF\xBF\xBD";
- else
- rep = "?";
- cr = ENC_CODERANGE_7BIT; +
- p = search_nonascii(p, e);
- if (!p) {
- p = e;
- }
- while (p < e) {
- int ret = rb_enc_precise_mbclen(p, e, enc);
- if (MBCLEN_CHARFOUND_P(ret)) {
- if ((unsigned char)*p > 127) cr = ENC_CODERANGE_VALID;
- p += MBCLEN_CHARFOUND_LEN(ret);
- }
- else if (MBCLEN_INVALID_P(ret)) {
- const char *q;
- long clen = rb_enc_mbmaxlen(enc);
- if (p > p1) rb_str_buf_cat(buf, p1, p - p1);
- q = RSTRING_END(buf); +
- if (e - p < clen) clen = e - p;
- if (clen < 3) {
- clen = 1;
- }
- else {

- long len = RSTRING_LEN(buf);
- clen--;
- rb_str_buf_cat(buf, p, clen);
- for (; clen > 1; clen--) {
- ret = rb_enc_precise_mbclen(q, q + clen, enc);
- if (MBCLEN_NEEDMORE_P(ret)) {
- break;
- }
- else if (MBCLEN_INVALID_P(ret)) {
- continue;
- }
- else {
- rb_bug("shouldn't reach here '%s'", q);
- }
- }
- rb_str_set_len(buf, len);
- }
- p += clen;
- p1 = p;
- rb_str_buf_cat2(buf, rep);
- p = search_nonascii(p, e);
- if (!p) {
- p = e;
- break;
- }
- }
- else if (MBCLEN_NEEDMORE_P(ret)) {
- break;
- }
- else {
- rb_bug("shouldn't reach here");
- }
- }
- if (p1 < p) {
- rb_str_buf_cat(buf, p1, p - p1);
- }
- if (p < e) {
- rb_str_buf_cat2(buf, rep);
- cr = ENC_CODERANGE_VALID;
- }
- ENCODING_CODERANGE_SET(buf, rb_enc_to_index(enc), cr);
- return buf;
- }
- else if (rb_enc_dummy_p(enc)) {
- return rb_str_dup(str);
- }
- else {
- /* ASCII incompatible */
- const char *p = RSTRING_PTR(str);
- const char *e = RSTRING_END(str);
- const char *p1 = p;
- /* 10 should be enough for the usual use case,
- * fixing a wrongly chopped character at the end of the string
- */
- long room = 10;
- VALUE buf = rb_str_buf_new(RSTRING_LEN(str) + room);
- const char *rep;
- long mbminlen = rb_enc_mbminlen(enc);
- static rb_encoding *utf16be;
- static rb_encoding *utf16le;
- static rb_encoding *utf32be;
- static rb_encoding *utf32le;
- if (!utf16be) {
- utf16be = rb_enc_find("UTF-16BE");
- utf16le = rb_enc_find("UTF-16LE");
- utf32be = rb_enc_find("UTF-32BE");
- utf32le = rb_enc_find("UTF-32LE");

```
}
if (enc == utf16be) {
rep = "\xFF\xFD";
}
else if (enc == utf16le) {
rep = "\xFD\xFF";
}
else if (enc == utf32be) {
rep = "\x00\x00\xFF\xFD";
}
else if (enc == utf32le) {
rep = "\xFD\xFF\x00\x00";
}
else {
rep = "?";
} +
while (p < e) {
int ret = rb_enc_precise_mbclen(p, e, enc);
if (MBCLEN_CHARFOUND_P(ret)) {
p += MBCLEN_CHARFOUND_LEN(ret);
}
else if (MBCLEN_INVALID_P(ret)) {
const char *q;
long clen = rb_enc_mbmaxlen(enc);
if (p > p1) rb_str_buf_cat(buf, p1, p - p1);
q = RSTRING_END(buf); +
if (e - p < clen) clen = e - p;
if (clen < mbminlen * 3) {
clen = mbminlen;
}
else {
long len = RSTRING_LEN(buf);
clen -= mbminlen;
rb_str_buf_cat(buf, p, clen);
for (; clen > mbminlen; clen-=mbminlen) {
ret = rb_enc_precise_mbclen(q, q + clen, enc);
if (MBCLEN_NEEDMORE_P(ret)) {
break;
}
else if (MBCLEN_INVALID_P(ret)) {
continue;
}
else {
rb_bug("shouldn't reach here '%s'", q);
}
}
rb_str_set_len(buf, len);
}
p += clen;
p1 = p;
rb_str_buf_cat2(buf, rep);
}
else if (MBCLEN_NEEDMORE_P(ret)) {
break;
}
else {
rb_bug("shouldn't reach here");
}
}
if (p1 < p) {
rb_str_buf_cat(buf, p1, p - p1);
}
if (p < e) {
rb_str_buf_cat2(buf, rep);
}
ENCODING_CODERANGE_SET(buf, rb_enc_to_index(enc), ENC_CODERANGE_VALID);
return buf;
```

- } +} + /***
  - Document-class: Symbol * @@ -7882,6 +8075,7 @@ Init_String(void) rb_define_method(rb_cString, "getbyte", rb_str_getbyte, 1); rb_define_method(rb_cString, "setbyte", rb_str_setbyte, 2); rb_define_method(rb_cString, "byteslice", rb_str_byteslice, -1);

- rb_define_method(rb_cString, "fix_invalid", rb_str_fix_invalid, 0);

  rb_define_method(rb_cString, "to_i", rb_str_to_i, -1);
  rb_define_method(rb_cString, "to_f", rb_str_to_f, 0);
  diff --git a/test/ruby/test_string.rb b/test/ruby/test_string.rb
  index 47f349c..2b0cfeb 100644
  --- a/test/ruby/test_string.rb
  +++ b/test/ruby/test_string.rb
  @@ -2031,6 +2031,29 @@ class TestString < Test::Unit::TestCase

  assert_equal(u("\x82")+("\u3042"*9), ("\u3042"*10).byteslice(2, 28))
  end
  +

- def test_fix_invalid

- assert_equal("\uFFFD\uFFFD\uFFFD", "\x80\x80\x80".fix_invalid)

- assert_equal("\uFFFDA", "\xF4\x80\x80A".fix_invalid)
  +

- # exapmles in Unicode 6.1.0 D93b

- assert_equal("\x41\uFFFD\uFFFD\x41\uFFFD\x41",

- [redacted]

- assert_equal("\x41\uFFFD\uFFFD\uFFFD\x41",

- [redacted]

- assert_equal("\u0061\uFFFD\uFFFD\uFFFD\u0062\uFFFD\u0063\uFFFD\uFFFD\u0064",

- [redacted]

  +

- assert_equal("abcdefghijklmnopqrstuvwxyz\u0061\uFFFD\uFFFD\uFFFD\u0062\uFFFD\u0063\uFFFD\uFFFD\u0064",

- [redacted]

  +

- assert_equal("\uFFFD\u3042".encode("UTF-16BE"),

- [redacted]

- [redacted]

- assert_equal("\uFFFD\u3042".encode("UTF-16LE"),

- [redacted]

- [redacted]

- end
  end

class TestString2 < TestString

=end

Related issues:

Related to Ruby master - Feature #6321: Find and repair bad bytes in encoding...	**Closed**	**04/19/2012**
Related to Ruby master - Bug #7967: String#encode invalid: :replace doesn't r...	**Rejected**	**02/26/2013**

Associated revisions

Revision 1e8a05c1 - 04/19/2013 08:21 PM - naruse (Yui NARUSE)

Add example for String#scrub

[Feature #6321] [Feature #6752] [Bug #7967]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40391 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 40391 - 04/19/2013 08:21 PM - naruse (Yui NARUSE)

Add example for String#scrub

[Feature #6321] [Feature #6752] [Bug #7967]

Revision 40391 - 04/19/2013 08:21 PM - naruse (Yui NARUSE)

Add example for String#scrub

[Feature #6321] [Feature #6752] [Bug #7967]

Revision 40391 - 04/19/2013 08:21 PM - naruse (Yui NARUSE)

Add example for String#scrub

[Feature #6321] [Feature #6752] [Bug #7967]

Revision 40391 - 04/19/2013 08:21 PM - naruse (Yui NARUSE)

Add example for String#scrub

[Feature #6321] [Feature #6752] [Bug #7967]

Revision 40391 - 04/19/2013 08:21 PM - naruse (Yui NARUSE)

Add example for String#scrub

[Feature #6321] [Feature #6752] [Bug #7967]

Revision 40391 - 04/19/2013 08:21 PM - naruse (Yui NARUSE)

Add example for String#scrub

[Feature #6321] [Feature #6752] [Bug #7967]

Revision eae1366b - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: suppress warnings

- string.c (rb_str_scrub): suppress maybe-uninitialized and empty body in an else-statement. [ruby-dev:45975] [Feature #6752]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40416 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 40416 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: suppress warnings

- string.c (rb_str_scrub): suppress maybe-uninitialized and empty body in an else-statement. [ruby-dev:45975] [Feature #6752]

Revision 40416 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: suppress warnings

- string.c (rb_str_scrub): suppress maybe-uninitialized and empty body in an else-statement. [ruby-dev:45975] [Feature #6752]

Revision 40416 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: suppress warnings

- string.c (rb_str_scrub): suppress maybe-uninitialized and empty body in an else-statement. [ruby-dev:45975] [Feature #6752]

Revision 40416 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: suppress warnings

- string.c (rb_str_scrub): suppress maybe-uninitialized and empty body in an else-statement. [ruby-dev:45975] [Feature #6752]

Revision 40416 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: suppress warnings

- string.c (rb_str_scrub): suppress maybe-uninitialized and empty body in an else-statement. [ruby-dev:45975] [Feature #6752]

Revision 40416 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: suppress warnings

- string.c (rb_str_scrub): suppress maybe-uninitialized and empty body in an else-statement. [ruby-dev:45975] [Feature #6752]

Revision 596ca948 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: fix for UTF-32

- string.c (rb_str_scrub): fix for UTF-32. strlen() on strings contain NUL returns wrong result, use sizeof operator instead. [ruby-dev:45975] [Feature #6752]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40417 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 40417 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: fix for UTF-32

- string.c (rb_str_scrub): fix for UTF-32. strlen() on strings contain NUL returns wrong result, use sizeof operator instead. [ruby-dev:45975] [Feature #6752]

Revision 40417 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: fix for UTF-32

- string.c (rb_str_scrub): fix for UTF-32. strlen() on strings contain NUL returns wrong result, use sizeof operator instead. [ruby-dev:45975] [Feature #6752]

Revision 40417 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: fix for UTF-32

- string.c (rb_str_scrub): fix for UTF-32. strlen() on strings contain NUL returns wrong result, use sizeof operator instead. [ruby-dev:45975] [Feature #6752]

Revision 40417 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: fix for UTF-32

- string.c (rb_str_scrub): fix for UTF-32. strlen() on strings contain NUL returns wrong result, use sizeof operator instead. [ruby-dev:45975] [Feature #6752]

Revision 40417 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: fix for UTF-32

- string.c (rb_str_scrub): fix for UTF-32. strlen() on strings contain NUL returns wrong result, use sizeof operator instead. [ruby-dev:45975] [Feature #6752]

Revision 40417 - 04/23/2013 02:58 AM - nobu (Nobuyoshi Nakada)

string.c: fix for UTF-32

- string.c (rb_str_scrub): fix for UTF-32. strlen() on strings contain NUL returns wrong result, use sizeof operator instead. [ruby-dev:45975] [Feature #6752]

History

#1 - 10/05/2012 09:47 AM - nobu (Nobuyoshi Nakada)

- Description updated

#2 - 10/05/2012 11:10 AM - kosaki (Motohiro KOSAKI)

今日のなるせさん、中田さんとのTwitter上での議論をもとにいくつかリクエスト（というか備忘録）

・個人的にはencode()よりも専用メソッド推し。理由は頻度。入力の正当性チェックなんてそこら中に需要あると思う
・メソッド名は replace_invalid_character みたいな思いっきり説明的な名前でいいような気がします。これをメソッドチェインしないでしょう。
あと、encode が invalid => replace なので用語合わせたほうがいい気がします。長すぎるなら replace_invalid で。
・オプショナル引数で置換文字(列)を変更できるようにしてほしい Unicode系でもU+FFFD がうれしくないケースは、ままありそう

#3 - 11/10/2012 05:23 PM - duerst (Martin Dürst)

- Target version changed from 2.0.0 to 2.6

I have thought about this a bit. Yui's patch to string treats this as a problem separat from transcoding. I think it is preferable to use the transcoding logic to implement this. The advantage is that exactly the same options and fallbacks can be used, and if we add a new option or fallback to transcode, it will be usable, too.

Some more notes: The checks for converting from one encoding to the same encoding are in str_transcode0. Anywhere else? We need some data to drive the conversion, but this should be easy to generate, and will be the same for many 8-bit encodings.

It will be easy to catch invalid byte sequences, but I'm not sure it's worth to check unassigned codepoints, at least not in Unicode.

I have changed the target version from 2.0.0 to next minor, because I don't think this will be ready for 2.0.0. But please change back if somebody can do it faster.

#4 - 01/09/2013 05:58 PM - knu (Akinori MUSHA)

=begin
(('+1')) for the functionality.

What we currently have (Encoding::Converter) is not enough to solve this problem because 1) it is mandatory to pick a different encoding to convert to for nothing, and even if you have decided to pick one 2) #primitive_errinfo does not give you offset information that is necessary to locate where the found invalid bytes are.

In addition to this proposal, I'd like String#encode to accept a proc instead of a fixed string as a :replace option to get a callback for each invalid byte so you can dynamically compose a replace string for the given invalid byte. Perl's encode() and decode() have this feature and it's very handy to investigate and visualize how a string is garbled. (e.g. (({replace: ->(byte) { "\e[7m<%02X>\e[m" % byte }})))

I don't have a particular opinion about the API, but having self-transcoders perform validation as duerst (Martin Dürst) implies sounds great to me if it could be properly implemented.

My tentative solution that is known to be very slow is here: ((URL:https://gist.github.com/4491446))
=end

#5 - 04/09/2013 04:38 AM - naruse (Yui NARUSE)

duerst (Martin Dürst) wrote:

> I have thought about this a bit. Yui's patch to string treats this as a problem separat from transcoding. I think it is preferable to use the transcoding logic to implement this. The advantage is that exactly the same options and fallbacks can be used, and if we add a new option or fallback to transcode, it will be usable, too.

This method doesn't need same options and fallbacks.
It need only invalid related, doesn't need undef related.
Moreover transcoder is usable only if Ruby has related transcoder of the target encoding.
But Ruby has some encodings which doesn't have transcoder for example emacs-mule.
Therefore this can't be built on transcoder.

> Some more notes: The checks for converting from one encoding to the same encoding are in str_transcode0. Anywhere else? We need some data to drive the conversion, but this should be easy to generate, and will be the same for many 8-bit encodings.

Yeah, I came to str_transcode0 and it is correct place.

The date we need is problem.
transcode doesn't have all the data though tool/transcode-tblgen.rb has some base data.
The only one which has all the data we need is enc/*.

> It will be easy to catch invalid byte sequences, but I'm not sure it's worth to check unassigned codepoints, at least not in Unicode.

If we need unassigned codepoints, we must define encodings more strictly.
Even if it is Unicode, it needs versions.
I don't think it's worth to check.

#6 - 04/09/2013 06:24 PM - naruse (Yui NARUSE)

I wrote a updated patch which include String#scrub and String#encode with extension.
String#scrub allows replacement as both argument or block.

diff --git a/string.c b/string.c
index 8b85739..bc973dc 100644
--- a/string.c
+++ b/string.c
@@ -7741,6 +7741,271 @@ rb_str_ellipsize(VALUE str, long len)
return ret;
}

+static VALUE
+str_compat_and_valid(VALUE str, rb_encoding *enc)
+{

- int cr;
- str = StringValue(str);
- cr = rb_enc_str_coderange(str);
- if (cr == ENC_CODERANGE_BROKEN) {
- rb_raise(rb_eArgError, "replacement must be valid byte sequence '%+"PRIsVALUE"'", str);
- }
- else if (cr == ENC_CODERANGE_7BIT) {
- rb_encoding *e = STR_ENC_GET(str);
- if (!rb_enc_asciicompat(enc)) {
- rb_raise(rb_eEncCompatError, "incompatible character encodings: %s and %s",
- rb_enc_name(enc), rb_enc_name(e));
- }
- }
- else { /* ENC_CODERANGE_VALID */
- rb_encoding *e = STR_ENC_GET(str);
- if (enc != e) {
- rb_raise(rb_eEncCompatError, "incompatible character encodings: %s and %s",
- rb_enc_name(enc), rb_enc_name(e));
- }
- }
- return str; +} + +/*
- * call-seq:
- * str.scrub -> new_str
- * str.scrub(repl) -> new_str
- * str.scrub{|bytes|} -> new_str
- *
- * If the string is invalid byte sequence then replace invalid bytes with given replacement
- * character, else returns self.
- */ +VALUE +rb_str_scrub(int argc, VALUE *argv, VALUE str) +{
- int cr = ENC_CODERANGE(str);
- rb_encoding *enc;
- VALUE repl; +
- if (cr == ENC_CODERANGE_7BIT || cr == ENC_CODERANGE_VALID)
- return rb_str_dup(str); +
- enc = STR_ENC_GET(str);
- rb_scan_args(argc, argv, "01", &repl);
- if (argc != 0) {
- repl = str_compat_and_valid(repl, enc);
- } +
- if (rb_enc_dummy_p(enc)) {
- return rb_str_dup(str);
- } +
- if (rb_enc_asciicompat(enc)) {
- const char *p = RSTRING_PTR(str);
- const char *e = RSTRING_END(str);
- const char *p1 = p;
- const char *rep;
- long replen;
- int rep7bit_p;
- VALUE buf = rb_str_buf_new(RSTRING_LEN(str));
- if (rb_block_given_p()) {
- rep = NULL;
- }
- else if (!NIL_P(repl)) {
- rep = RSTRING_PTR(repl);
- replen = RSTRING_LEN(repl);
- rep7bit_p = (ENC_CODERANGE(repl) == ENC_CODERANGE_7BIT);
- }

- else if (enc == rb_utf8_encoding()) {
- rep = "\xEF\xBF\xBD";
- replen = strlen(rep);
- rep7bit_p = FALSE;
- }
- else {
- rep = "?";
- replen = strlen(rep);
- rep7bit_p = TRUE;
- }
- cr = ENC_CODERANGE_7BIT; +
- p = search_nonascii(p, e);
- if (!p) {
- p = e;
- }
- while (p < e) {
- int ret = rb_enc_precise_mbclen(p, e, enc);
- if (MBCLEN_NEEDMORE_P(ret)) {
- break;
- }
- else if (MBCLEN_CHARFOUND_P(ret)) {
- cr = ENC_CODERANGE_VALID;
- p += MBCLEN_CHARFOUND_LEN(ret);
- }
- else if (MBCLEN_INVALID_P(ret)) {
- /*
- * p1~p: valid ascii/multibyte chars
- * p ~e: invalid bytes + unknown bytes
- */
- long clen = rb_enc_mbmaxlen(enc);
- if (p > p1) {
- rb_str_buf_cat(buf, p1, p - p1);
- } +
- if (e - p < clen) clen = e - p;
- if (clen <= 2) {
- clen = 1;
- }
- else {
- const char *q = p;
- clen--;
- for (; clen > 1; clen--) {
- ret = rb_enc_precise_mbclen(q, q + clen, enc);
- if (MBCLEN_NEEDMORE_P(ret)) break;
- else if (MBCLEN_INVALID_P(ret)) continue;
- else UNREACHABLE;
- }
- }
- if (rep) {
- rb_str_buf_cat(buf, rep, replen);
- if (!rep7bit_p) cr = ENC_CODERANGE_VALID;
- }
- else {
- repl = rb_yield(rb_enc_str_new(p1, clen, enc));
- repl = str_compat_and_valid(repl, enc);
- rb_str_buf_cat(buf, RSTRING_PTR(repl), RSTRING_LEN(repl));
- if (ENC_CODERANGE(repl) == ENC_CODERANGE_VALID)
- cr = ENC_CODERANGE_VALID;
- }
- p += clen;
- p1 = p;
- p = search_nonascii(p, e);
- if (!p) {
- p = e;
- break;
- }
- }
- else {
- UNREACHABLE;
- }
- }
- if (p1 < p) {
- rb_str_buf_cat(buf, p1, p - p1);
- }
- if (p < e) {
- if (rep) {

- rb_str_buf_cat(buf, rep, replen);
- if (!rep7bit_p) cr = ENC_CODERANGE_VALID;
- }
- else {
- repl = rb_yield(rb_enc_str_new(p, e-p, enc));
- repl = str_compat_and_valid(repl, enc);
- rb_str_buf_cat(buf, RSTRING_PTR(repl), RSTRING_LEN(repl));
- if (ENC_CODERANGE(repl) == ENC_CODERANGE_VALID)
- cr = ENC_CODERANGE_VALID;
- }
- }
- ENCODING_CODERANGE_SET(buf, rb_enc_to_index(enc), cr);
- return buf;
- }
- else {
- /* ASCII incompatible */
- const char *p = RSTRING_PTR(str);
- const char *e = RSTRING_END(str);
- const char *p1 = p;
- VALUE buf = rb_str_buf_new(RSTRING_LEN(str));
- const char *rep;
- long replen;
- long mbminlen = rb_enc_mbminlen(enc);
- static rb_encoding *utf16be;
- static rb_encoding *utf16le;
- static rb_encoding *utf32be;
- static rb_encoding *utf32le;
- if (!utf16be) {
- utf16be = rb_enc_find("UTF-16BE");
- utf16le = rb_enc_find("UTF-16LE");
- utf32be = rb_enc_find("UTF-32BE");
- utf32le = rb_enc_find("UTF-32LE");
- }
- if (!NIL_P(repl)) {
- rep = RSTRING_PTR(repl);
- replen = RSTRING_LEN(repl);
- }
- else if (enc == utf16be) {
- rep = "\xFF\xFD";
- replen = strlen(rep);
- }
- else if (enc == utf16le) {
- rep = "\xFD\xFF";
- replen = strlen(rep);
- }
- else if (enc == utf32be) {
- rep = "\x00\x00\xFF\xFD";
- replen = strlen(rep);
- }
- else if (enc == utf32le) {
- rep = "\xFD\xFF\x00\x00";
- replen = strlen(rep);
- }
- else {
- rep = "?";
- replen = strlen(rep);
- } +
- while (p < e) {
- int ret = rb_enc_precise_mbclen(p, e, enc);
- if (MBCLEN_NEEDMORE_P(ret)) {
- break;
- }
- else if (MBCLEN_CHARFOUND_P(ret)) {
- p += MBCLEN_CHARFOUND_LEN(ret);
- }
- else if (MBCLEN_INVALID_P(ret)) {
- const char *q = p;
- long clen = rb_enc_mbmaxlen(enc);
- if (p > p1) rb_str_buf_cat(buf, p1, p - p1); +
- if (e - p < clen) clen = e - p;
- if (clen <= mbminlen * 2) {
- clen = mbminlen;
- }
- else {
- clen -= mbminlen;

- for (; clen > mbminlen; clen-=mbminlen) {
- ret = rb_enc_precise_mbclen(q, q + clen, enc);
- if (MBCLEN_NEEDMORE_P(ret)) break;
- else if (MBCLEN_INVALID_P(ret)) continue;
- else UNREACHABLE;
- }
- }
- if (rep) {
- rb_str_buf_cat(buf, rep, replen);
- }
- else {
- repl = rb_yield(rb_enc_str_new(p, e-p, enc));
- repl = str_compat_and_valid(repl, enc);
- rb_str_buf_cat(buf, RSTRING_PTR(repl), RSTRING_LEN(repl));
- }
- p += clen;
- p1 = p;
- }
- else {
- UNREACHABLE;
- }
- }
- if (p1 < p) {
- rb_str_buf_cat(buf, p1, p - p1);
- }
- if (p < e) {
- if (rep) {
- rb_str_buf_cat(buf, rep, replen);
- }
- else {
- repl = rb_yield(rb_enc_str_new(p, e-p, enc));
- repl = str_compat_and_valid(repl, enc);
- rb_str_buf_cat(buf, RSTRING_PTR(repl), RSTRING_LEN(repl));
- }
- }
- ENCODING_CODERANGE_SET(buf, rb_enc_to_index(enc), ENC_CODERANGE_VALID);
- return buf;
- } +} + /**
    - Document-class: Symbol * @@ -8222,6 +8487,7 @@ Init_String(void) rb_define_method(rb_cString, "getbyte", rb_str_getbyte, 1); rb_define_method(rb_cString, "setbyte", rb_str_setbyte, 2); rb_define_method(rb_cString, "byteslice", rb_str_byteslice, -1);

- rb_define_method(rb_cString, "scrub", rb_str_scrub, -1);

  rb_define_method(rb_cString, "to_i", rb_str_to_i, -1);
  rb_define_method(rb_cString, "to_f", rb_str_to_f, 0);
  diff --git a/test/dl/test_callback.rb b/test/dl/test_callback.rb
  index 8ae652b..ef24235 100644
  --- a/test/dl/test_callback.rb
  +++ b/test/dl/test_callback.rb
  @@ -61,9 +61,11 @@ module DL
  addr = set_callback(TYPE_VOID, 1) do |foo|
  called = true
  end

- [redacted]

  func = CFunc.new(addr, TYPE_VOID, 'test')
  f = Function.new(func, [TYPE_VOIDP])

- [redacted]

  f.call(nil)

  assert called, 'function should be called'
  diff --git a/test/ruby/test_m17n.rb b/test/ruby/test_m17n.rb
  index a8d56a4..60834bb 100644
  --- a/test/ruby/test_m17n.rb
  +++ b/test/ruby/test_m17n.rb
  @@ -1489,4 +1489,38 @@ class TestM17N < Test::Unit::TestCase
  s.untrust
  assert_equal(true, s.b.untrusted?)
  end
  +

- def test_scrub
- assert_equal("\uFFFD\uFFFD\uFFFD", u("\x80\x80\x80").scrub)
- assert_equal("\uFFFDA", u("\xF4\x80\x80A").scrub)
  +
- # exapmles in Unicode 6.1.0 D93b
- assert_equal("\x41\uFFFD\uFFFD\x41\uFFFD\x41",
- 
- assert_equal("\x41\uFFFD\uFFFD\uFFFD\x41",
- 
- assert_equal("\u0061\uFFFD\uFFFD\uFFFD\u0062\uFFFD\u0063\uFFFD\uFFFD\u0064",
- 
- assert_equal("abcdefghijklmnopqrstuvwxyz\u0061\uFFFD\uFFFD\uFFFD\u0062\uFFFD\u0063\uFFFD\uFFFD\u0064",
- 

  +
- assert_equal("\u3042\u3013", u("\xE3\x81\x82\xE3\x81").scrub("\u3013"))
- assert_raise(Encoding::CompatibilityError){ u("\xE3\x81\x82\xE3\x81").scrub(e("\xA4\xA2")) }
- assert_raise(TypeError){ u("\xE3\x81\x82\xE3\x81").scrub(1) }
- assert_raise(ArgumentError){ u("\xE3\x81\x82\xE3\x81\x82\xE3\x81").scrub(u("\x81")) }
- assert_equal(e("\xA4\xA2\xA2\xAE"), e("\xA4\xA2\xA4").scrub(e("\xA2\xAE")))
  +
- assert_equal("\u3042", u("\xE3\x81\x82\xE3\x81").scrub{|x|'<'+x.unpack('H*')[0]+'>'})
- assert_raise(Encoding::CompatibilityError){ u("\xE3\x81\x82\xE3\x81").scrub{e("\xA4\xA2")} }
- assert_raise(TypeError){ u("\xE3\x81\x82\xE3\x81").scrub{1} }
- assert_raise(ArgumentError){ u("\xE3\x81\x82\xE3\x81\x82\xE3\x81").scrub{u("\x81")} }
- assert_equal(e("\xA4\xA2\xA2\xAE"), e("\xA4\xA2\xA4").scrub{e("\xA2\xAE")})
  +
- assert_equal("\uFFFD\u3042".encode("UTF-16BE"),
- 
- 
- assert_equal("\uFFFD\u3042".encode("UTF-16LE"),
- 
- 
- end
  end
  diff --git a/transcode.c b/transcode.c
  index de12c04..9c940ed 100644
  --- a/transcode.c
  +++ b/transcode.c
  @@ -2652,6 +2652,8 @@ str_transcode_enc_args(VALUE str, volatile VALUE *arg1, volatile VALUE *arg2,
  return dencidx;
  }

+VALUE rb_str_scrub(int argc, VALUE *argv, VALUE str);
+
static int

str_transcode0(int argc, VALUE *argv, VALUE *self, int ecflags, VALUE ecopts)
{
@@ -2686,6 +2688,17 @@ str_transcode0(int argc, VALUE *argv, VALUE *self, int ecflags, VALUE ecopts)
ECONV_XML_ATTR_CONTENT_DECORATOR|
ECONV_XML_ATTR_QUOTE_DECORATOR)) == 0) {
if (senc && senc == denc) {

- if (ecflags & ECONV_INVALID_MASK) {
- if (!NIL_P(ecopts)) {
- VALUE rep = rb_hash_aref(ecopts, sym_replace);
- dest = rb_str_scrub(1, &rep, str);
- }
- else {
- dest = rb_str_scrub(0, NULL, str);
- }
- *self = dest;
- return dencidx;
- }   return NIL_P(arg2) ? -1 : dencidx;   }   if (senc && denc && rb_enc_asciicompat(senc) && rb_enc_asciicompat(denc)) {

#7 - 04/10/2013 08:04 PM - naruse (Yui NARUSE)

というわけで、[ruby-dev:47241] の通り patch を更新し、提案を以下の通り更新しましたので、まつもとさんコメントいただけませんか。

= 不正なバイト列なStringを綺麗にするやつ

== ここまでのあらすじ
趣旨は認められつつも、String#encode を使う API 案と独自名の API 案が出てまとまらなかった。

== 本提案の概要

- API は String#encoode を用いるものと新メソッドの2つを提供する
- 新メソッドの名称は String#scrub とする

== API

API は String#encoode を用いるものと新しいメソッドである String#scrub の2つを提供する。

== String#encode

本 API が用意されるべき理由は、iconv からの連想でこの API を用いて本用途が実現されると
期待する人が多いからである。
一方で、String#encode は CSI な変換エンジンを持つ Ruby では引数が複雑になりがちである。
例えば、典型的な UTF-8 の文字列を修復したい場合、

str.encode("UTF-8", invalid: :replace)

置換文字列を指定したい場合には

str.encode("UTF-8", invalid: :replace, replace: "\u3013")

などとなります。
これにさらにブロックを加えてより複雑な fallback を実現したくなるわけが、
Ruby M17N では CSI な多段変換を用いているので、一つのブロックで fallback を
実現しようとすると複雑になってしまうため、従来よりそのような機能は見送ってきた。
具体的には、ブロックが呼ばれる際には、

- invalid か undef か
- from encoding (多段の途中では思いもよらぬエンコーディングのことがある)
- to encoding (多段の途中では思いもよらぬエンコーディングのことがある) というようなパラメータが存在する。そのため、1つのブロックに詰め込むには複雑にすぎると思われる。

== String#scrub

本用途に特化した、よりシンプルな API を提供すると同時に、
ブロックを用いた fallback 等の高度な操作をも可能とする新 API 。
名前は zpool scrub より。
http://docs.oracle.com/cd/E19253-01/819-6260/gbbxi/

=== str.scrub

Unicode 系ならば U+FFFD (Replacement Character) を置換文字とし、
それ以外の場合は ? を置換文字とする。

=== str.scrub("**")

指定した文字列を置換文字とする。

=== str.scrub{|x| '<'+x.ord.to_s(16)+'>' }

ブロック引数として不正なバイト列を与え、引数を置換文字とする。

#8 - 04/19/2013 12:31 PM - matz (Yukihiro Matsumoto)

OK, I agree with enhancement of String#encoding and String#scrub.

Matz.

#9 - 04/20/2013 05:21 AM - naruse (Yui NARUSE)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r40391.
Yui, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

Add example for String#scrub

[Feature #6321] [Feature #6752] [Bug #7967]