Ruby master - Bug #680

csv.rb: CSV.parse is too late when encoding is mismatch
10/24/2008 10:00 AM - xibbar (Takeyuki FUJIOKA)

| Status:          | Closed               |
| Priority:        | Normal               |
| Assignee:        | JEG2 (James Gray)    |
| Target version:  | 2.0.0                |
| ruby -v:         | Backport:            |

Description

=begin
I think this result is true, but encoding mismatch raise is too late.

see:
% time ruby19 -rcsv -e 'CSV.parse("\x82\xA0,\x82\xA2\n"*10000).force_encoding("shift_jis")'
ruby19 -rcsv -e 0.30s user 0.02s system 96% cpu 0.330 total
% time ruby19 -rcsv -e 'CSV.parse("\x82\xA0,\x82\xA2\n"*10000))'
/Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1981:in =~': broken UTF-8 string (ArgumentError)
from /Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1981:in init_separators'
from /Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1563:in initialize'
from /Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1350:in new'
from /Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1350:in parse'
from -e:1
ruby19 -rcsv -e 'CSV.parse("\x82\xA0,\x82\xA2\n"*10000))' 1.55s user 2.57s system 90% cpu 4.530 total
=end

History

#1 - 10/24/2008 03:54 PM - duerst (Martin Dürst)
=begin
A default for the source encoding has been discussed quite a long
time ago (in some Japanese meetings or on ruby-dev, I don't remember),
and the conclusion was that the source encoding has to be given
(with a magic comment) in the file itself (unless the file is all ascii).

The reason for this is that the source encoding is a property of the
source, and nothing else. On very simple scripts, it might occasionally
be slightly easier if it were the same as default_external or
default_internal, but this is only the case as long as you stay
in exactly the same environment, and don't move the script.
But scripts grow and move, so it's better to get the settings
right at the start.

However, as far as I remember, the idea was that for -e,
default_external should be used, because that's what one
is using in a shell. I'm not sure why this doesn't work below.
(assuming Takeyuki is working in a Shift_JIS environment,
which isn't completely sure).

Regards,    Martin.

At 12:12 08/10/24, Michael Selig wrote:

    Hi,

    This bug actually brings up an interesting issue - should the source
    encoding default to something other than UTF-8 (ie: if it is not specified
    in the "magic comment")?

    Perhaps it should default to the encoding specified by the user's locale?
    Or perhaps it should default to the value of "default_internal" if it is
    set? Or even default_external?
I suggest that it should default to "default_internal" if that is set, and then to the locale encoding if not.

What do others think? Having it default to the locale in this case would probably avoid the encoding mismatch entirely (and the resulting confusion).

Cheers
Mike

On Fri, 24 Oct 2008 11:58:33 +1100, Takeyuki Fujioka

redmine@ruby-lang.org wrote:

Bug #680: csv.rb: CSV.parse is too late when encoding is mismatch
http://redmine.ruby-lang.org/issues/show/680

Author: Takeyuki Fujioka
Status: Open, Priority: Normal
Category: lib, Target version: 1.9.x

I think this result is true, but encoding mismatch raise is too late.

see:
% time ruby19 -rcsv -e
'CSV.parse("(\x82\xA0,\x82\xA2\n"*10000).force_encoding("shift_jis")")'
ruby19 -rcsv -e 0.30s user 0.02s system 96% cpu 0.330 total

% time ruby19 -rcsv -e 'CSV.parse("(\x82\xA0,\x82\xA2\n"*10000))")
/Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1981:in =~': broken UTF-8 string (ArgumentError)
from /Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1981:in init_separators'
from /Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1563:in initialize'
from /Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1350:in new'
from /Users/fujioka/local/lib/ruby/1.9.0/csv.rb:1350:in parse'
from -e:1:in '
ruby19 -rcsv -e 'CSV.parse("(\x82\xA0,\x82\xA2\n"*10000))")  1.55s user 2.57s system 90% cpu 4.530 total

http://redmine.ruby-lang.org

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

#2 - 10/24/2008 05:06 PM - matz (Yukihiro Matsumoto)
=begin
Hi,

In message "Re: [ruby-core:19473] Re: Default source encoding (Was: [Bug #680] csv.rb: CSV.parse is too late when encoding is mismatch)" on Fri, 24 Oct 2008 16:48:04 +0900, "Michael Selig" michael.selig@fs.com.au writes:

|The problem I am trying to solve is the compatibility of string literals
|in your source and strings from other sources.
|"default_internal" was introduced to try to make all strings the same
|encoding to avoid incompatibilities. But at the moment string literals
|seem to default to the source encoding or to UTF-8 if it is not set
|please correct me if I am wrong). What I was suggesting was a way to make
|string literals be compatible.
You are correct here.

This normally isn't a problem if:
(a) All string literals are 7 bit ASCII, or
(b) The source encoding matches "default_internal"

If the source encoding of a program containing non-ascii string literals
is set different from default_internal, you are asking for trouble, and
would defeat the purpose of default_internal. Therefore to prevent the
programmer from having to remember to specify both, it makes sense to me
that the source encoding should default to default_internal. I think this
is important.

The point is that when we have a source code written in source
encoding, the literals naturally encoded in that encoding. So do we
need to convert string literals in to default encoding? But
conversion can bring us more troubles, since they tend to change the
meaning, for example what \[ - ] mean, where \ and \ are
multi byte characters and their corresponding codepoints (and sorting
order) differ in converted encoding?

(By the way, I am not talking about libraries here. As I have stressed
previously, libraries should be carefully written to either use ASCII
string literals only, or to make sure that it transcodes them properly.)

That makes me feel much better, so we can limit the issue about the
scripts only.

Finally, are you suggesting that "-e" should perform differently to a
single-line ruby script? That seems non-intuitive to me.

-e takes programs from command line shell, which probably yields
strings in locale encoding anyway. But we cannot assume that for
scripts contained in files.

---

By the way, I am not talking about libraries here. As I have stressed
previously, libraries should be carefully written to either use ASCII
string literals only, or to make sure that it transcodes them properly.)

That makes me feel much better, so we can limit the issue about the
scripts only.

Finally, are you suggesting that "-e" should perform differently to a
single-line ruby script? That seems non-intuitive to me.

-e takes programs from command line shell, which probably yields
strings in locale encoding anyway. But we cannot assume that for
scripts contained in files.

---

File sample.csv added

Please save as 'sample.csv' attached file.
This file include japanese UTF-8 string in first line.
Other line is us-ascii. Line number count is 5001.

% time ruby19 -Eutf-8 -rcsv -e 'CSV.parse(open("sample.csv","r").read)'
0.23s user 0.01s system 96% cpu 0.254 total

this is OK very fast.

But:

% time ruby19 -Eeuc-jp -rcsv -e 'CSV.parse(open("sample.csv","r").read)'
3.93s user 6.38s system 98% cpu 10.457 total

this result is very slow.
I hope raise as soon as encoding mismatch found.

# Sorry, I don't understand M17N's default_external and default_internal behavior.
I can't reply about M17N's problem.

---

#4 - 10/25/2008 01:01 AM - nobu (Nobuyoshi Nakada)

Hi,

At Fri, 24 Oct 2008 23:00:27 +0900, James Gray wrote in [ruby-core:19481]:

I work on TextMate and we use Ruby all over the place inside that application. I'm sure we have hundreds of scripts in there. We try hard to make sure everything in TextMate is UTF-8, so now we get errors out of Ruby 1.9. To fix, we need to add hundreds of magic comments and worse, train our users who often write their own automations in Ruby why they have to do the same to make their code work.

The real issue here is that you can argue the user doesn't even know the proper encoding these scripts should be using. Only TextMate really knows the encoding it's going to hand-off the data in.

Though I don't know about TextMate at all, ruby-mode.el in 1.9 deals with magic comments automatically.

--
Nobu Nakada

---

#5 - 10/25/2008 09:57 AM - JEG2 (James Gray)

- Status changed from Open to Closed
- % Done changed from 0 to 100

Applied in changeset r19931.

---

#6 - 10/25/2008 09:58 AM - JEG2 (James Gray)

- Assignee set to JEG2 (James Gray)

Thanks for finding the bug in my logic. It should be much faster now:

```
$ time ruby_dev -Eeuc-jp -rlib/csv -e 'CSV.parse(open("/Users/james/Desktop/sample.csv","r").read)'
 from /Users/james/Documents/ruby_source/trunk/lib/csv.rb:1350:in new'
 from /Users/james/Documents/ruby_source/trunk/lib/csv.rb:1350:in parse'
 from -e:1:in '
real   0m0.053s
user   0m0.039s
sys    0m0.011s
```

---

#7 - 10/26/2008 03:26 PM - nobu (Nobuyoshi Nakada)

---
Hi,

At Sun, 26 Oct 2008 11:25:58 +0900, Michael Selig wrote in [ruby-core:19515]:

   1) My preference would be to always encode string literals constructed with "x.." as ASCII-8BIT, ignoring the source encoding. This means that if you really want to use such a literal as an encoded string, you must use "force_encoding". I think this would be much clearer and get rid of the "ambiguity".

   2) My suggestion for "defaulting" the source encoding was an attempt to avoid having to do this (but probably not a good way!). It isn't a big deal, and I understand the argument that the source encoding is a property of the script. My original suggestion (last month) of a special magic comment was to have a way of specifying BOTH the default_internal and source encoding once, but this idea was rejected.

I'd prefer to default the internal encoding to the source encoding of the main script.

   3) Perhaps this check could be based on the library's source encoding? If this were done, most libraries would have to use a source encoding of US-ASCII (or just have no encoding magic comment) not UTF-8, so that non-Unicode default_internal's will work. Perhaps Ruby could be smarter, and only flag an error if there actually is an incompatible string literal in the library?

What about comments? I suspect it might not be a good idea.

   4) Also it means that:
      ruby test.rb
      may perform differently than:
      ruby -e "cat test.rb"

magic comments are effective with -e too.

   $ ruby19 -e 'p ENCODING'
   Encoding:EUC-JP
   $ ruby19 -e '#-- encoding: utf-8 --' -e 'p ENCODING'
   Encoding:UTF-8

Therefore no differences if the file has the magic comment.

--
Nobu Nakada

=end

#8 - 10/26/2008 09:34 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,
At Sun, 26 Oct 2008 17:20:17 +0900, Michael Selig wrote in [ruby-core:19518]:

I'd prefer to default the internal encoding to the source encoding of the main script.

But then how do you tell Ruby NOT to set "default_internal"?

I think defaulting the internal encoding to something other is bad.

It also means that comments must be in the default_internal encoding (see your comment below).

I don't follow you here, all comments should be written in the source encoding. Why default_internal affects?

Therefore no differences if the file has the magic comment.

That's true, but my point was "why should a simple non-m17n non-ascii ruby program have to contain the magic comment"?

Because, non-ascii. It's definitely enough reason.

--
Nobu Nakada
=end

#9 - 10/27/2008 02:08 PM - nobu (Nobuyoshi Nakada)
=begin

Hi,

At Mon, 27 Oct 2008 07:28:42 +0900, Michael Selig wrote in [ruby-core:19525]:

Yes you are right, and I was not suggesting doing that.
But Matz wants to default default_internal to nil. With your proposal, how do you do that and still set the source encoding?

I don't like the idea setting default_internal from source encoding, but meant "it feels less worse" by "prefer".

My original suggestion was to use an extended "magic comment" to set both.

But it can't keep the source encoding unset, and "internal_encoding" has no effect for Emacs.

Isn't backward compatibility with 1.8 scripts more important?
You are now forcing anyone with 1.8 scripts containing non-ascii string literals to put in a magic comment, otherwise you get "invalid multibyte char (US-ASCII)" error in 1.9.

In other words, what you want is -K option?

--
Nobu Nakada
=end
Hi,


OK, I don't use Emacs, and no one told me that before, thanks! I assumed it would work, but I admit I didn't test it.

Then is there another form of magic comment that can be used - eg:

"internal encoding: XXXX" or "encoding: XXXX internal" that does work with Emacs?

No. Magic comments without -*- markers are for VIM, like

# vim: set encoding=UTF-8

and, both of VIM and Emacs wouldn't work with your examples.

What I am saying is that we need to consider backward compatibility of Ruby scripts. James Grey brought up an example with his "Textmate scripts" which contain UTF-8 multibyte string literals, which used to work with 1.8, but do not in 1.9, because they need either a "magic comment" or, as you say "-KU". Either way, 1.9 is not truly backward compatible when it comes to simple, non-m17n, non-ascii scripts, because you have to either modify the script or add a flag to the ruby options. There must be lots of Japanese ruby scripts which will have a similar issue.

Even in 1.8 or prior, -Ks has been mandatory for Shift_JIS sources, so they have had -K in the shebang lines already.

Defaulting source encoding to locale encoding (like -e does) should fix this (as long as the end-user's locale is correct), right?

Yes if they match.

I guess if necessary James can put "-KU" in the RUBYOPT environment variable to save having to add multiple magic comments, but I feel this shouldn't be necessary.

-U option would be better.

--
Nobu Nakada

Even in 1.8 or prior, -Ks has been mandatory for Shift_JIS sources, so they have had -K in the shebang lines already.
Why then can I write a ruby 1.8 script which does a "puts" of a Shift_JIS string (no shebang or magic comment), and have it run fine without -Ks?

Because you are avoiding troublesome chars. Without such chars, we can't write the words "display", "table", "software" and "ruby".

I guess if necessary James can put "-Ku" in the RUBYOPT environment variable to save having to add multiple magic comments, but I feel this shouldn't be necessary.

-U option would be better.

I don't think that will work:

t2.rb is a single line script which does a puts of a short UTF-8 multibyte string.

Indeed. -U sets only internal encoding, whereas -Ku sets also external and source encodings. Therefore -U isn't direct replacement for -Ku.

But it's very ambiguous and dangerous to imply encodings. We can't trust locale for this purpose, at least.

You can use BOM to mean that the source is written in UTF-8.

--
Nobu Nakada
=end

#12 - 10/27/2008 07:38 PM - duerst (Martin Dürst)

<=begin

At 07:28 08/10/27, Michael Selig wrote:

I thought one of your points was that you would like to be able to write
Japanese (or other non-asci) comments which is otherwise only ascii
(which may use "\u" in literals, and want default_internal to be UTF-8).

This means that the source encoding should be Japanese. Your suggestion of
defaulting default_internal to the source encoding means that it will be
set to Japanese. I am not sure that this is always desirable. (This is
very minor - you can always override it)

I'm not sure what you mean by "Japanese". It's no problem at all
to use UTF-8 to write Japanese. And I guess if somebody uses
\u literals and wants default_internal to be UTF-8, they'll
in most cases use UTF-8 for the source encoding (comments or
whatever else).

If you mean Japanese legacy encodings (such as Shift_JIS and
EUC-JP), then your are correct, but it would be very rare
for somebody to use Shift_JIS or EUC-JP for comments when
the program is otherwise supposed to run all-UTF-8.

Isn't backward compatibility with 1.8 scripts more important?
You are now forcing anyone with 1.8 scripts containing non-ascii string
literals to put in a magic comment, otherwise you get "invalid multibyte
char (US-ASCII)" error in 1.9.
Well, yes, that's actually the point of it. Wherever necessary, get everybody to declare their encoding. It may be somewhat suboptimal in the transition phase, but after that, we know what we're dealing with.

Regards, Martin.

---

Michael Selig wrote:

I am not sure why you would want to keep the source encoding unset when setting default_internal at the top of a script. Perhaps you could explain.

The simplest case is a script in US-ASCII only, but where you want the data to be handled e.g. in UTF-8.

Regards, Martin.

---

James Gray wrote:

They sure could, yeah. Our policy for TextMate development has always been that UTF-8 is king. We use it heavily and I'm sure some scripts do contain multibyte characters in UTF-8.

Wouldn't it be only these scripts (including those that contain \x escapes for UTF-8) that need the encoding indication at the top? (please note that literals with \u escapes are automatically UTF-8).

Regards, Martin.

---

Michael Selig wrote:

On Mon, 27 Oct 2008 20:55:32 +1100, Nobuyoshi Nakada nobu@ruby-lang.org wrote:

Hi,

At Mon, 27 Oct 2008 15:57:03 +0900, Michael Selig wrote in [ruby-core:19535]:

Even in 1.8 or prior, -Ks has been mandatory for Shift_JIS sources, so they have had -K in the shebang lines already.

Why then can I write a ruby 1.8 script which does a "puts" of a
Shift_JIS
string (no shebang or magic comment), and have it run fine without -Ks?

Because you are avoiding troublesome chars. Without such
chars, we can't write the words "display", "table", "software"
and "ruby".

OK, I'm sure you know more about Japanese encodings that I do.

To give you the details, these characters, in Shift_JIS, are
encoded with two bytes, the second of which is the same byte
as e.g. a backslash.

But my original point is that 1.8 scripts exist which contain multibyte
characters (eg UTF-8) which work fine under 1.8 without-K, but will fail
under 1.9 unless a magic comment or -K is provided.

Yes, that's because 1.8 is essentially garbage-in-garbage out.
If you are careful about certain bytes, you can essentially have
arbitrary byte sequences in your script, and Ruby 1.8 won't complain.

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 - 10/27/2008 09:07 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Mon, 27 Oct 2008 19:17:45 +0900,
Michael Selig wrote in [ruby-core:19540]:

But my original point is that 1.8 scripts exist which contain multibyte
characters (eg UTF-8) which work fine under 1.8 without-K, but will fail
under 1.9 unless a magic comment or -K is provided.

It just seemed working by chance.

But it's very ambiguous and dangerous to imply encodings. We
can't trust locale for this purpose, at least.

It's a trade-off between that and backward compatibility. I think the
"danger" is not high and it gives backward compatibility, so my vote would
be to use it.

And it will suddenly crash or behave weirdly by moving other
locales.

Anyway, I think I understand the needs to specify source
encoding without magic comments. Is the option for that
purpose an acceptable solution?

--
Nobu Nakada
=begin
#17 - 10/27/2008 09:12 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Mon, 27 Oct 2008 19:37:58 +0900, Martin Duerst wrote in [ruby-core:19541]:

    If you mean Japanese legacy encodings (such as Shift_JIS and EUC-JP), then your are correct, but it would be very rare for somebody to use Shift_JIS or EUC-JP for comments when the program is otherwise supposed to run all-UTF-8.

I don't do it of course, but know that some people love to do it.

--
Nobu Nakada
=end

#18 - 10/28/2008 01:16 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:19550] Re: String literal encoding (Was: Default source encoding (Was: [Bug #680] csv.rb: CSV.parse is too late when encoding is mismatch))" on Tue, 28 Oct 2008 00:12:46 +0900, James Gray james@grayproductions.net writes:

|I wasn't aware -KU still worked though, as Michael pointed out. I ignored now.
|It may be that the TextMate team could use that. What all does it set
|in 1.9? Source encoding obviously. It seems to affect
|default_external as well, but not touch default_internal. Do I have
|that right? Does it have any other special effects?

-Ku (or -KU) specifies to

* default script encoding to be UTF-8
* default_external encoding to be UTF-8 unless it's specified previously by -E or -U
* do not touch default_internal

Will -KU stay supported for the foreseeable future?

Yes.

matz.
=end

#19 - 10/31/2008 06:39 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,


    Anyway, I think I understand the needs to specify source encoding without magic comments. Is the option for that purpose an acceptable solution?

Here is the patch to add options:

    --encoding=external:internal:source
    --external-encoding=enc
--- internal-encoding=enc
--- source-encoding=enc

Index: ruby.c

--- ruby.c (revision 20075)
+++ ruby.c (working copy)
@@ -623,5 +623,5 @@ dump_option(const char *str, int len, vo
static void
-set_internal_encoding_once(struct cmdline_options *opt, const char *e, int elen)
+set_option_encoding_once(const char *type, VALUE *name, const char *e, int elen)
{
    VALUE ename;
@@ -630,27 +630,16 @@ set_internal_encoding_once(struct cmdlin
        ename = rb_str_new(e, elen);
    if (opt->intern.enc.name &&
        rb_funcall(ename, rb_intern("casecmp"), 1, opt->intern.enc.name) != INT2FIX(0)) {
        if (*name &&
            rb_funcall(ename, rb_intern("casecmp"), 1, *name) != INT2FIX(0)) { rb_raise(rb_eRuntimeError,
            "default internal already set to %s", RSTRING_PTR(opt->intern.enc.name));
            "%s already set to %s", type, RSTRING_PTR("name"));   }
        opt->intern.enc.name = ename;
        "name = ename;"
    }

-void
-set_external_encoding_once(struct cmdline_options *opt, const char *e, int elen)
+void
+set_option_encoding_once(const char *type, VALUE *name, const char *e, int elen)
{
    VALUE ename; -
    if (!elen) elen = strlen(e);
    ename = rb_str_new(e, elen); -
    if (opt->ext.enc.name &&
        rb_funcall(ename, rb_intern("casecmp"), 1, opt->ext.enc.name) != INT2FIX(0)) {
        rb_raise(rb_eRuntimeError,
            "default external already set to %s", RSTRING_PTR(opt->ext.enc.name));
    }
    opt->ext.enc.name = ename; -
    
+static
+void
+set_internal_encoding_once(struct cmdline_options *opt, const char *e, int elen)
+
+{ static
+    VALUE ename; -
+    if (telen) ename = strlenc(e);
+    ename = rb_str_new(e, elen); -
+    if (opt->ext.enc.name &&
+        rb_funcall(ename, rb_intern("casecmp"), 1, opt->ext.enc.name) != INT2FIX(0)) {
+        rb_raise(rb_eRuntimeError,
+            "default external already set to %s", RSTRING_PTR(opt->ext.enc.name));
+    }
+    opt->ext.enc.name = ename; -
+    
+    set_option_encoding_once("default internal", &opt->intern.enc.name, e, elen) +
+    set_option_encoding_once("default external", &opt->ext.enc.name, e, elen)
+
+static int
@@ -956,13 +945,29 @@ proc_options(int argc, char **argv, stru
    encoding:
        p = strchr(s, ':');
        if (p) {
            if (p > s)
                set_external_encoding_once(opt, s, p-s);
            if (*++p)
                set_internal_encoding_once(opt, p, 0);
            }
        }
    }

    set_external_encoding_once(opt, s, 0);
    do { +# define set_encoding_part(type) 
        if (!(p = strchr(s, ':'))) { 
            set_encoding_part(external);
            if (!"type"(s = ++p)) break;
            set_encoding_part(source); +# undef set_encoding_part
        } while (0);
    }

    else if (is_option_with_arg("internal-encoding", Qfalse, Qtrue)) {
        set_internal_encoding_once(opt, s, 0);
    }

09/20/2021
else if (is_option_with_arg("external-encoding", Qfalse, Qtrue)) {
    set_external_encoding_once(opt, s, 0);
} else if (is_option_with_arg("source-encoding", Qfalse, Qtrue)) {
    set_source_encoding_once(opt, s, 0);
} else if (strcmp("version", s) == 0) {

--
Nobu Nakada
=end

#20 - 10/31/2008 06:59 PM - nobu (Nobuyoshi Nakada)
=begin
Hi,

At Fri, 31 Oct 2008 18:38:24 +0900,
Nobuyoshi Nakada wrote in [ruby-core:19655]:

+#define set_internal_encoding_once(opt, e, elen) 
    set_option_encoding_once("default_internal", &opt->intern.enc.name, e, elen)
+#define set_external_encoding_once(opt, e, elen) 
    set_option_encoding_once("default_external", &opt->ext.enc.name, e, elen)

Sorry, missed these 2 lines.

#define set_source_encoding_once(opt, e, elen) 
set_option_encoding_once("source", &opt->src.enc.name, e, elen)

--
Nobu Nakada
=end

#21 - 10/31/2008 07:06 PM - duerst (Martin Dürst)
=begin
At 18:38 08/10/31, Nobuyoshi Nakada wrote:

Hi,

At Mon, 27 Oct 2008 21:07:16 +0900,
Nobuyoshi Nakada wrote in [ruby-core:19546]:

Anyway, I think I understand the needs to specify source
encoding without magic comments. Is the option for that
purpose an acceptable solution?

Here is the patch to add options:

Great work!

--encoding=external:internal:source
--external-encoding=enc
--internal-encoding=enc
--source-encoding=enc

I personally don't like the last one, and the :source in the first
one, but I guess there are situations where they can be very helpful
(e.g. testing with different encodings).

I also think that it would be good to have the values of --encoding
and -E look/work the same, so unless :source already works on -E,
I think having just --source-encoding for the case that the
source encoding must be set by an option should be okay.
This will also make it easier to distinguish in documentation
that --source-encoding is really only for very special occasions,
and declaring the source encoding in the script itself is strongly
preferred.

Regards, Martin.
Hi,

At Fri, 31 Oct 2008 19:05:25 +0900, Martin Duerst wrote in [ruby-core:19657]:

```
--encoding=external:internal:source
--external-encoding=enc
--internal-encoding=enc
--source-encoding=enc
```

I personally don't like the last one, and the :source in the first one, but I guess there are situations where they can be very helpful (e.g. testing with different encodings).

I also think that it would be good to have the values of --encoding and -E look/work the same, so unless :source already works on -E, I think having just --source-encoding for the case that the source encoding must be set by an option should be okay.

-E equals to --encoding.

This will also make it easier to distinguish in documentation that --source-encoding is really only for very special occasions, and declaring the source encoding in the script itself is strongly preferred.

Since these four options are separated, so it's easy to remove some of them.

--
Nobu Nakada

Files
sample.csv  97.7 KB  10/24/2008  xibbar (Takeyuki FUJIOKA)