Ruby master - Feature #14077

Add Encoding::FILESYSTEM and Encoding::LOCALE constants

11/03/2017 04:35 PM - nirvdrum (Kevin Menard)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

The Encoding class has constants defined for all of the standard encodings, but does not have constants defined for some of the special encodings. In particular, the "filesystem" and "locale" encodings do not have a corresponding Encoding::FILESYSTEM and Encoding::LOCALE. As a result, every time they need to be used they must be looked up using Encoding.find. As far as I can tell, neither of these special encodings can change after the Ruby process has been started up. Even changing the $LANG environment variable within the Ruby process doesn't seem to affect the value for these special encodings.

Therefore, I'm proposing the introduction of Encoding::FILESYSTEM and Encoding::LOCALE constants both as a matter of consistency and as a matter of efficiency.

History

#1 - 11/03/2017 05:11 PM - rrroybbbean (RRRoy BBBean)

On Fri, 2017-11-03 at 16:35 +0000, ruby@kevin.nirvdrum.com wrote:

> Issue #14077 has been reported by nirvdrum (Kevin Menard).
> Feature #14077: Add Encoding::FILESYSTEM and Encoding::LOCALE
> constants
> https://bugs.ruby-lang.org/issues/14077
>
> - Author: nirvdrum (Kevin Menard)
> - Status: Open
> - Priority: Normal
> - Assignee:
> - Target version: ... Therefore, I'm proposing the introduction of Encoding::FILESYSTEM and Encoding::LOCALE constants both as a matter of consistency and as a matter of efficiency. This is a great idea. It makes Ruby programming easier and more intuitive. It's small innovations like this that have made Ruby into an awesome programming experience. Do more work in less time with less code.

#2 - 11/03/2017 06:34 PM - shevegen (Robert A. Heiler)

I am in agreement with the feature-suggestion. Not sure whether
it should be a constant or a method or both but I agree that it
may be useful to have direct support for this in ruby.

> Even changing the $LANG environment variable within the Ruby
> process doesn't seem to affect the value for these special
> encodings.

You mean you used ENV['LANG']? Manipulating ENV works for me
for other variables at the least such as CFLAGS and so forth.

Perhaps there may be a reason why there is no direct support,
nobu may perhaps know more - they probably sleep right now in
japan at 03:32 ... :)

rrroybbbean wrote:

> It's small innovations like this that have made Ruby into
> an awesome programming experience.

Yes - please do not forget the documentation for it though!
Since others rely on documentation.

Matz said several times that one (core?) part of ruby's philosophy
is the "human aspect" aka how something is used with ruby. I think

> that this is also a reason why the ruby core team often likes to see "real world use cases" to determine how/if something is used.

#3 - 11/03/2017 09:00 PM - nirvdrum (Kevin Menard)

shevegen (Robert A. Heiler) wrote:

> > Even changing the $LANG environment variable within the Ruby process doesn't seem to affect the value for these special encodings.
>
> You mean you used ENV['LANG']? Manipulating ENV works for me for other variables at the least such as CFLAGS and so forth.
>
> Perhaps there may be a reason why there is no direct support, nobu may perhaps know more - they probably sleep right now in japan at 03:32 ... :)

Sorry if I wasn't clear. Both of these encoding values are dependent on the system locale at the time the process is started. They're not not hard-coded into Ruby like UTF-8 or ASCII-8BIT. My point was that while altering $LANG before the process starts can affect the value of these special encodings, once the process is loaded the special encoding values never change, even if you change the $LANG value from within the process. I think what Ruby is doing is the preferable behavior. But, since the special encodings never change after process start up, I think there should be a faster way to reference them (e.g., a constant).

#4 - 11/03/2017 09:31 PM - rrroybbbean (RRRoy BBBean)

On Fri, 2017-11-03 at 18:34 +0000, shevegen@gmail.com wrote:

> Issue #14077 has been updated by shevegen (Robert A. Heiler).
> I am in agreement with the feature-suggestion. Not sure whether it should be a constant or a method or both but I agree that it may be useful to have direct support for this in ruby.
> ...
> Matz said several times that one (core?) part of ruby's philosophy is the "human aspect" aka how something is used with ruby. I think that this is also a reason why the ruby core team often likes to see "real world use cases" to determine how/if something is used.

Many of my cheesy ruby scripts manipulate directory hierarchies on both windows and linux, often to fix problems that occur when you share an NTFS-formatted external disk drive between systems.

This is one of the most frequent things that I have to do, since many of my files (and some directories) use Korean UTF-8 characters:

Dir.entries('/.../mydir/',).each do |base|

I know that I must specify the encoding on Windows 7, or else it assumes Windows-1252 and messes up multi-byte characters. This code also works fine on Ubuntu 14, Fedora 24 and Debian 9, although I don't even know what the default or filesystem encoding is on Linux systems.

FYI, on Windows 7, I work exclusively with NTFS and Fat32. On Linux, I routinely work with EXT4, NTFS and Fat32. Are you aware that the NTFS driver for Linux allows you to create filesystem objects with names that are unworkable under Windows? [names with embedded colons : for instance]

I had to go to the Internet to figure out that I needed to use :encoding=>'UTF-8' to properly handle multi-byte characters on Windows

1. It would have been nice to have Ruby tell me what the default encodings were. That's a lame reason for inclusion of this proposed feature, but it's all I have at the moment.

In the past, I ran into another problem, where I found embedded text of a character type different than the enclosing text. I find that even today, in filenames and text that mix English, Japanese and Korean texts into a single string or file. I blame word-processors for this mess. I used to jump through hoops to handle the problem, then I got smart and just forced the encoding to UTF-8, replacing bad characters

with ". In this situation, I don't see how knowing the filesystem or default encodings would help, since the person who created the Frankenstein-text didn't realize what they were doing.