

Ruby - Bug #7267

Dir.glob on Mac OS X returns unexpected string encodings for unicode file names

11/02/2012 07:54 PM - kennygrant (Kenny Grant)

Status:	Closed	Backport:
Priority:	Normal	
Assignee:	duerst (Martin Dürst)	
Target version:	2.6	
ruby -v:	ruby 1.9.3p194 (2012-04-20 revision 35410) [x86_64-darwin11.4.0]	
Description		
Tested on Ruby 1.9.3-p194 and ruby-2.0.0-preview1 on Mac OS X 10.7.5		
When calling file system methods with Ruby on Mac OS X, it is not possible to manipulate the resulting file name as a normal UTF-8 string, even though it reports the encoding as UTF-8. It seems to be a UTF-8-MAC string, even when the default encoding is set to UTF-8. This leads to confusion as the string can be manipulated normally except for any unicode characters, which seem to be decomposed. So a regexp using utf-8 characters won't work on the string, unless it is first converted from UTF-8-MAC. I'd expect the string encoding to be UTF-8, or at least to report that it is not a normal UTF-8 string if it has to be UTF-8-MAC for some reason. Example, run with a file called Testé.txt in the same folder: <pre>def transform_string s puts "Testing string #{s}" puts s.gsub(/é/, 'TEST') end Dir.glob("./*.txt").each do f puts "Inline string works as expected" s = "./Testé.txt" puts transform_string s puts "File name from Dir.glob does not" puts transform_string f puts "Encoded file name works as expected, though it is reported as UTF-8, not UTF-8-MAC" f.encode!('UTF-8','UTF-8-MAC') puts transform_string f end</pre>		
Related issues:		
Related to Ruby - Bug #2154: filesystem encoding of UNIX		Rejected
Related to Ruby - Feature #7280: How to set filesystem encoding		Closed
Related to Ruby - Feature #10084: Add Unicode String Normalization to String ...		Closed

Associated revisions

Revision 29dc980e6562b4083f72f121465eca2f5427af39 - 03/19/2013 08:06 AM - nobu (Nobuyoshi Nakada)

compose HFS file names

- dir.c (glob_helper): compose HFS file names from UTF8-MAC.
[ruby-core:48745] [Bug #7267]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@39821 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 29dc980e - 03/19/2013 08:06 AM - nobu (Nobuyoshi Nakada)

compose HFS file names

- dir.c (glob_helper): compose HFS file names from UTF8-MAC.
[ruby-core:48745] [Bug #7267]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@39821 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 1891b60f268780fe4e9d970ffa759c3d7dd0e00b - 05/20/2013 06:11 PM - nobu (Nobuyoshi Nakada)

dir.c: compose HFS file names

- dir.c (dir_each): compose HFS file names from UTF8-MAC. [ruby-core:48745] [Bug #7267]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40868 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 1891b60f - 05/20/2013 06:11 PM - nobu (Nobuyoshi Nakada)

dir.c: compose HFS file names

- dir.c (dir_each): compose HFS file names from UTF8-MAC. [ruby-core:48745] [Bug #7267]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@40868 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 11/02/2012 08:32 PM - kennygrant (Kenny Grant)

- *File results.txt added*

Output of the test.rb script:

Tested on Ruby 2.0.0-preview and 1.9.3 on Mac OS X
1.9.3x86_64-darwin11.4.0
Inline string works as expected
Source encoding: UTF-8
External encoding: UTF-8
Name: `"/Testé.txt"`
Encoding: UTF-8
Chars: `[".", "/", "T", "e", "s", "t", "é", ".", "t", "x", "t"]`
Codepoints: [46, 47, 84, 101, 115, 116, 233, 46, 116, 120, 116]
Bytes: [46, 47, 84, 101, 115, 116, 195, 169, 46, 116, 120, 116]
Testing string `"/Testé.txt"`
`"/TestTEST.txt"`

File name from Dir.glob does not
Source encoding: UTF-8
External encoding: UTF-8
Name: `"/Testé.txt"`
Encoding: UTF-8
Chars: `[".", "/", "T", "e", "s", "t", "e", "t", "é", ".", "t", "x", "t"]`
Codepoints: [46, 47, 84, 101, 115, 116, 101, 769, 46, 116, 120, 116]
Bytes: [46, 47, 84, 101, 115, 116, 101, 204, 129, 46, 116, 120, 116]
Testing string `"/Testé.txt"`
`"/Testé.txt"`

Source encoding: UTF-8
External encoding: UTF-8
Name: `"/Testé.txt"`
Encoding: UTF-8
Chars: `[".", "/", "T", "e", "s", "t", "é", ".", "t", "x", "t"]`
Codepoints: [46, 47, 84, 101, 115, 116, 233, 46, 116, 120, 116]
Bytes: [46, 47, 84, 101, 115, 116, 195, 169, 46, 116, 120, 116]
Testing string `"/Testé.txt"`
`"/TestTEST.txt"`

#2 - 11/03/2012 03:54 AM - meta (mathew murphy)

Relevant links:

<http://search.cpan.org/~tomita/Encode-UTF8Mac-0.03/lib/Encode/UTF8Mac.pm>

Seems to me Ruby should pick one of the standard normalization forms for all UTF-8 data, and convert when necessary.

Apparently there are OS X library calls to assist:

http://developer.apple.com/library/mac/qa/qa1235/_index.html

#3 - 11/03/2012 04:18 AM - shyouhei (Shyouhei Urabe)

meta (mathew murphy) wrote:

Seems to me Ruby should pick one of the standard normalization forms

No, Ruby's M17N design is that it never picks one standard form of strings. Strings have various encodings and it's you, not ruby, who should pick a right thing.

I admit this is not the one only solution for this mess. Other languages like Perl have different opinions. But the way ruby works is this. So please, do it for yourself.

And I admit it should be much more easier for you to do the choice. We need some brush-up.

#4 - 11/03/2012 07:50 AM - kennygrant (Kenny Grant)

- *File writer.rb* added

The problem I encountered here was that although the encoding is also UTF-8, apparently there are several flavours of UTF-8, NFC or NFD. This is not something most Ruby users will be familiar with, and I'm not sure it's really the same as having to choose your own encoding and always convert to and from it (which I understand is necessary). I was very confused at first as these strings all report UTF-8 encoding, yet behave differently - they display in the same way but act differently in some circumstances because of the underlying bytes.

Writing out to a file with a UTF-8 string name (composed) will result in getting a different UTF-8 string (decomposed) when read back in later with ruby Dir.glob or other File/Dir methods, so apparently there is automatic translation one way but not another. The file name read back in appears to work until you try matching against the string or manipulating the parts, so it leads to silent failures where code which worked for ascii strings will fail on names which are not plain ascii on Mac OS X, unless you explicitly reconvert the file name to a composed form again.

What I would expect to happen is for Ruby file system methods to convert back to composed form on reading in file names again, so that matches on a strings or regexp defined as UTF-8 would work correctly. As it is these fail, and the string displays normally but does not behave as you would expect.

Apologies if this has already been considered and I am rehashing an old argument, I just found this behaviour somewhat puzzling until I worked out what it was doing, and even then it is painful to have to consider which flavour of UTF-8 is in use and convert to NFC all the time.

A related bug is that if you match a glob on the exact name, you will receive a UTF-8 string which is NFC, whereas if you try a partial match, you will receive NFD, so the behaviour can be inconsistent. See attached file for an example which writes out a name then tries to match it straight afterward.

It would be great if Ruby could just consistently return NFC (as is used when you use UTF-8) and convert as necessary for the file system, but never expose that to the user.

Thanks for your time.

#5 - 11/03/2012 01:27 PM - shyouhei (Shyouhei Urabe)

Just another reason why Unicodes sucks.

Anyway, I know your feeling and I wish I could help you. The problem is, the world isn't built on top of Mac OS. So there're virtually thousands of different formats of hard disks, with different ways of storing filenames. In *your* mac, it might be some sort of normalized Unicode. Not the case for others, even on a mac, like when you network-mount a Linux-hosted logical volume.

So it's not a matter of normalizing. The real problem is we can't, in practice, know the real encoding of a filename. There's simply no way to obtain an encoding of a path. We have to ASSUME what it is instead. And all you're familiar with the fact that assumptions always sucks. Yes, this is the reason of the mess. Creating a file with UTF-8 filename and resulting a filename in different kind of UTF-8, isn't because Ruby's being evil. It's the default behaviour of your filesystem. We just can't know about that situation. Can't.

PS. See also how MacFUSE project thinks this exact same issue as "WontFix": <http://code.google.com/p/macfuse/issues/detail?id=139>

PS. I repeat I don't believe the current situation is the best. We should have a better workaround. Don't know how though.

#6 - 11/04/2012 12:43 AM - kennygrant (Kenny Grant)

Thanks for the explanation. I didn't think Ruby was being evil :)

If the translation from UTF8-MAC -> UTF-8 is entirely non-lossy and would do no harm to other UTF-8 strings (these decomposed patterns would only occur with the intention of displaying an accent wouldn't they?), and Ruby is set to use UTF-8 for all encodings, perhaps the right thing to do here would be to auto-translate UTF8-MAC to UTF-8 on reading all file names assumed to be UTF-8 on Mac OS, as the OS default is decomposed, but the default in Ruby is composed. I can't think of a situation when anyone would want to use UTF8-MAC in a ruby script as opposed to UTF-8, and if they did presumably they could explicitly convert to it, and if the file system was set to use composed anyway, this translation would not affect the name. The string is coming to me as UTF-8 from Dir.glob, so at that stage it knows (or rather has assumed) it is UTF-8 (though in fact it seems it is UTF-8-MAC), and presumably at that stage it could do a conversion to be sure it was canonical UTF-8 before returning the string, IF the encoding was set to UTF-8 already?

Hopefully this would not affect any other users/file systems, but I'm afraid I don't know enough to make that judgement call and may well have overlooked something. I'd be happy to try submitting a patch but have not done so before and would likely hinder rather than help as this is a big complex issue with lots of potential side-effects, so this is really just a long-term suggestion from an end user's point of view.

As you say, it would be nice to have a cleaner solution to this at some point, so I hope one can be found which would get rid of this potentially confusing behaviour for those using UTF-8 on Mac OS X, and not cause more work for those using other encodings or operating systems.

#7 - 11/05/2012 04:59 PM - duerst (Martin Dürst)

On 2012/11/03 3:54, meta (mathew murphy) wrote:

Issue [#7267](#) has been updated by meta (mathew murphy).

Relevant links:

<http://search.cpan.org/~tomita/Encode-UTF8Mac-0.03/lib/Encode/UTF8Mac.pm>

Seems to me Ruby should pick one of the standard normalization forms for all UTF-8 data, and convert when necessary.

That wouldn't work because we want Ruby to be able to work on different normalizations (because there is different data out there, or data in different forms has to be produced).

Regards, Martin.

Apparently there are OS X library calls to assist:

http://developer.apple.com/library/mac/qa/qa1235/_index.html

There are quite a few implementations of these in pure Ruby, too. That's not the problem. The problem is figuring out where and when to apply them.

Regards, Martin.

#8 - 11/08/2012 11:39 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to duerst (Martin Dürst)

Martin-sensei, should we do anything for this issue?
If not, could you close the ticket?

Thanks,

--

Yusuke Endoh mame@tsg.ne.jp

#9 - 11/09/2012 05:59 PM - naruse (Yui NARUSE)

What I would expect to happen is for Ruby file system methods to convert back to composed form on reading in file names again, so that matches on a strings or regexp defined as UTF-8 would work correctly. As it is these fail, and the string displays normally but does not behave as you would expect.

An issue is people may write decomposed filename.

A imaginary use case is a program which make a filename from the name of a music output from iTunes.

iTunes manages texts with UTF8-MAC.

So the people will confuse.

Apologies if this has already been considered and I am rehashing an old argument, I just found this behaviour somewhat puzzling until I worked out what it was doing,

This issue is discussed for long time.

First Ruby 1.9.0 set strings derived from filenames UTF8-MAC.

But some reported that if filenames is UTF8-MAC, it is hard to compare with normal UTF-8 strings.

So from 1.9.2 filenames become UTF-8.

I find there is no correct simple answer.

and even then it is painful to have to consider which flavour of UTF-8 is in use and convert to NFC all the time.

More painful thing is normal UTF-8 is not NFCed UTF-8.

There is both NFCed one and NFDed one.

People are living with such ambiguity all the time even if they don't notice.

It would be great if Ruby could just consistently return NFC (as is used when you use UTF-8) and convert as necessary for the file system, but never expose that to the user.

Again Ruby can't always return NFC strings because filenames are not normalized and may contain both NFCed one and NFDed one on other than HFS+.
(you may know Mac OS X's default filesystem is HFS+)

If the translation from UTF8-MAC -> UTF-8 is entirely non-lossy and would do no harm to other UTF-8 strings

Yes until all part of the converting string is truly UTF8-MAC.

perhaps the right thing to do here would be to auto-translate UTF-8-MAC to UTF-8 on reading all file names assumed to be UTF-8 on Mac OS, as the OS default is decomposed, but the default in Ruby is composed.

I slightly doubt that this feature truly make people happy and there are no side effect even if without technical difficulty.

Hopefully this would not affect any other users/file systems, but I'm afraid I don't know enough to make that judgement call and may well have overlooked something.

On Mac OS X, there are other than HFS+.
Mac OS X can use UFS, CDFS, NFS and so on.
They don't normalize filenames.

If you NFCed such filenames, you lost the file.
Moreover if you mount filesystems, a path may contain names from different filesystems like
/ - HFS+
/foo - UFS
/foo/bar - ext4 over NFS
/foo/bar/baz - NTFS
/foo/bar/baz/cd - CDFS

Here, only "foo" is normalized.
If bar is "e" (decomposed) and a directory named "è" (composed) in the parent directory, the path lost the file.

#10 - 11/09/2012 05:59 PM - naruse (Yui NARUSE)

2012/11/3 meta (mathew murphy) meta@pobox.com:

Relevant links:

<http://search.cpan.org/~tomita/Encode-UTF8Mac-0.03/lib/Encode/UTF8Mac.pm>

Seems to me Ruby should pick one of the standard normalization forms for all UTF-8 data, and convert when necessary.

Apparently there are OS X library calls to assist:
http://developer.apple.com/library/mac/qa/qa1235/_index.html

Ruby already has UTF8-MAC to UTF-8 converter.
You can use it by `str.encode("UTF-8", "UTF8-MAC")`.

--

NARUSE, Yui naruse@airemix.jp

#11 - 11/09/2012 07:30 PM - kennygrant (Kenny Grant)

Thanks for the comments on this issue. I'm not clear on what the UTF8-MAC encoding represents, are there docs on this Ruby behaviour and the problems involved somewhere?

At present `Dir.glob` has inconsistent behaviour even working with the same file/filesystem:

It may return a filename marked UTF-8 which is NFD, or NFC, depending on the glob pattern you call it with (see `writer.rb` attachment to this issue). That's a small issue though and just indicates a wider complex problem.

An issue is people may write decomposed filename. A imaginary use case is a program which make a filename from the name of a music output

from iTunes. iTunes manages texts with UTF8-MAC. So the people will confuse.

OK, so in this case someone is unwittingly using a mix of UTF-8 NFC (any strings they create in ruby with legible accents) and UTF-8 NFD (any strings they get from iTunes say) in their script, which could lead to issues even before writing file names. If they get NFD from iTunes, then try to match on a track name with a regexp, it won't work unless they convert to NFC or explicitly create an NFD string will it? So even ignoring the file system there are issues here with labelling two different normalization forms UTF-8 as it leads to naive expectations of compatibility which are false.

More painful thing is normal UTF-8 is not NFCed UTF-8. There is both NFCed one and NFDed one.

Yes, it might be good to make this very clear in the Ruby docs, as so many people use UTF-8 for all strings now, probably mostly without understanding this issue (I'm aware not everyone in the Ruby community uses UTF-8 for other reasons). I certainly didn't know about it and it took quite a while to find any docs about it at all (most of which were for perl or other languages). Thanks for taking the time to explain it, and sorry if I missed some obvious notes in the docs for the ruby string class say.

One thing I don't understand though, is that you say there are both in normal use - in use of Ruby ignoring file systems, if you create a string or regexp, NFC is the default isn't it? So Ruby has chosen one default for UTF-8 strings created in Ruby (as it must), but has to interact with lots of systems which might or might not be using NFC. At present we seem to have a de-facto default normalization of NFC, but nothing is translated to it when it comes from the OS. That might be a very hard problem, but in principle it would be nice to have one normalization blessed as the default so that all strings in a given encoding are comparable. The results of leaving them as they are supplied are really unexpected, and people using Ruby are not going to want to manually convert every string they touch from outside Ruby to NFC in case it was touched by HFS or created as NFD.

First Ruby 1.9.0 set strings derived from filenames UTF8-MAC.

But some reported that if filenames is UTF8-MAC, it is hard to compare with normal UTF-8 strings.

This is interesting as it's exactly the behaviour I expected (if it's not possible to cleanly translate to NFC) - if strings are coming through as UTF-8 NFD, I'd expect them to be marked as such somehow (for example by being marked as encoding UTF8-MAC) - is there any indication? Then at least it is clear that they are not comparable or compatible with the NFC ruby strings I get when creating a string `s = "détente"`. Removing the explicit encoding doesn't really solve the problem of strings being hard to compare, it just hides it until comparisons fail. By default, Ruby seems to use NFC UTF-8, which is what I'd expect, so I also expected to either receive strings which are comparable directly, or for those strings to be marked as some other encoding/normalization if they come from an HFS+ file system, so that I can deal with them even if Ruby doesn't. Is the normalization exposed somewhere on strings? The current situation was surprising and unexpected, though now that it has been explained I see why it might occur and how hard it is to fix.

In the example code attached to this issue, I've used `str.encode!("UTF-8", "UTF8-MAC")` to translate file names/paths to NFC which works for my uses, but from naruse's comments, it seems that would fail in certain circumstances?

If the translation from UTF8-MAC -> UTF-8 is entirely non-lossy and would do no harm to other UTF-8 strings
Yes until all part of the converting string is truly UTF8-MAC.

I assumed from others' comments that UTF8-MAC was purely a sub-encoding used to indicate the use of decomposed strings, but would appreciate some more detail (if anyone has a link) on what exactly it involves, and if translation from UTF8-MAC to UTF8 can lose information that implies other differences. If the only difference is the decomposition (patterns which do not occur in NFC), I'd expect re-encoding to be idempotent and not affect NFC strings and thus harmless to apply to NFC strings or strings containing a mix. Re the file-system example, I had assumed that if you ask HFS to write to a file on a mounted file system HFS would normalize all names to NFD (as it does for any HFS files), but perhaps that is incorrect.

I suppose the above boils down to this question:

Is there a correct way to handle this situation, and never fail when comparing a default Ruby string (NFC) against a file from any file system which may be NFD?

#12 - 11/09/2012 09:53 PM - naruse (Yui NARUSE)

2012/11/9 kennygrant (Kenny Grant) kennygrant@gmail.com:

Thanks for the comments on this issue. I'm not clear on what the UTF8-MAC encoding represents, are there docs on this Ruby behaviour and the problems involved somewhere?

see several lines at the end of `enc/utf_8.c`.

It may return a filename marked UTF-8 which is NFD, or NFC, depending on the glob pattern you call it with (see `writer.rb` attachment to this issue). That's a small issue though and just indicates a wider complex problem.

`writer.rb`'s two puts output the same result.
What do you mean?

An issue is people may write decomposed filename. A imaginary use case is a program which make a filename from the name of a music output from iTunes. iTunes manages texts with UTF8-MAC. So the people will confuse.

OK, so in this case someone is unwittingly using a mix of UTF-8 NFC (any strings they create in ruby with legible accents) and UTF-8 NFD (any strings they get from itunes say) in their script, which could lead to issues even before writing file names. If they get NFD from itunes, then try to match on a track name with a regexp, it won't work unless they convert to NFC or explicitly create an NFD string will it?

It will work unless the regexp highly depends composed string.

One thing I don't understand though, is that you say there are both in normal use - in use of Ruby ignoring file systems, if you create a string or regexp, NFC is the default isn't it?

No, NFC is not default.

The fact is that many IMEs outputs composed characters.

Once a decomposed characters is mixed in a string, the character lives as is.

It won't normalized.

So Ruby has chosen one default for UTF-8 strings created in Ruby (as it must), but has to interact with lots of systems which might or might not be using NFC. At present we seem to have a de-facto default normalization of NFC, but nothing is translated to it when it comes from the OS. That might be a a very hard problem, but in principle it would be nice to have one normalization blessed as the default so that all strings in a given encoding are comparable. The results of leaving them as they are supplied are really unexpected, and people using Ruby are not going to want to manually convert every string they touch from outside Ruby to NFC in case it was touched by HFS or created as NFD.

Ruby don't normalize characters.

It treat them as they are.

Windows, Linux, and other file systems also don't normalize.

Moreover NFC/NFD lost information.

If a filename is decomposed characters on Windows or Linux, NFC for the filename lost it.

First Ruby 1.9.0 set strings derived from filenames UTF8-MAC.

But some reported that if filenames is UTF8-MAC, it is hard to compare with normal UTF-8 strings.

This is interesting as it's exactly the behaviour I expected (if it's not possible to cleanly translate to NFC) - if strings are coming through as UTF-8 NFD, I'd expect them to be marked as such somehow (for example by being marked as encoding UTF8-MAC) - is there any indication?

A no so simple point is UTF8-MAC string is valid as UTF-8.

Then at least it is clear that they are not comparable or compatible with the NFC ruby strings I get when creating a string s = "détente".

Even if the string is accidentally composed, there are no guarantee that a string is always composed.

If the translation from UTF8-MAC -> UTF-8 is entirely non-lossy and would do no harm to other UTF-8 strings
Yes until all part of the converting string is truly UTF8-MAC.

I assumed from others' comments that UTF8-MAC was purely a sub-encoding used to indicate the use of decomposed strings, but would appreciate some more detail (if anyone has a link) on what exactly it involves, and if translation from UTF8-MAC to UTF8 can lose information that implies other differences. If the only difference is the decomposition (patterns which do not occur in NFC), I'd expect re-encoding to be idempotent and not affect NFC strings and thus harmless to apply to NFC strings or strings containing a mix. Re the file-system example, I had assumed that if you ask HFS to write to a file on a mounted file system HFS would normalize all names to NFD (as it does for any HFS files), but perhaps that is incorrect.

A UTF-8 string is not always NFCed.

I suppose the above boils down to this question:

Is there a correct way to handle this situation, and never fail when comparing a default Ruby string (NFC) against a file from any file system which may be NFD?

No way.

And again, Ruby string is not NFC.

--

NARUSE, Yui naruse@airemix.jp

#13 - 11/09/2012 10:56 PM - kennygrant (Kenny Grant)

see several lines at the end of enc/utf_8.c.

Thanks for the links, I'll read them to get a better understanding of this, I'm sorry I can't read the other two tickets associated with this as that would probably be enlightening too. I suspect the problem comes because of my somewhat parochial focus on using Ruby in English and other roman languages, where the text editors and other tools tend to generate NFC UTF-8, so I expected to get NFC UTF-8 back. When I type

```
s = "détente"
```

the string is composed, which is what I'd expect, it isn't accidental. If there was some way in future to ask Ruby to use only composed forms when interacting with file systems, it would be really useful for use with western languages on Mac OS.

writer.rb's two puts output the same result.

Here is the result I get from writer.rb (with a few more tests)

(ruby 1.9.3p194 (2012-04-20 revision 35410) [x86_64-darwin11.4.0], Mac OS X 10.7.5 HFS+ disk, I saw this on 2.0 too)- one result is decomposed, one is composed, depending on the string passed in.

After writing out a file with an NFC name to disk (resulting in an NFD filename on disk presumably).

GLOB on "./*.txt" pattern - result NFD

```
[".", "/", "t", "e", "s", "t", "e", "", ".", "t", "x", "t"]
```

GLOB on "./testé.txt" - result NFC

```
[".", "/", "t", "e", "s", "t", "é", ".", "t", "x", "t"]
```

GLOB on "./testé.*" - fails, no match

GLOB on "./teste*" - match result NFD

```
[".", "/", "t", "e", "s", "t", "e", "", ".", "t", "x", "t"]
```

No way. And again, Ruby string is not NFC.

OK thanks, I'll just have to accept that converting to composed to match my internal ruby strings (which are NFC) is necessary when dealing with an HFS file system.

#14 - 02/19/2013 12:11 AM - mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 2.6

Should we close this ticket?

--

Yusuke Endoh mame@tsg.ne.jp

#15 - 03/18/2013 03:22 PM - naruse (Yui NARUSE)

Attaching my proof of concept code.

```
diff --git a/dir.c b/dir.c
index d4b3dd3..126c27e 100644
--- a/dir.c
+++ b/dir.c
@@ -81,6 +81,84 @@ char strchr(char, char);

#define rb_sys_fail_path(path) rb_sys_fail_str(path)

+#if defined(APPLE)
+
+#include <sys/param.h>
+#include <sys/mount.h>
+
```

```
+static int
+is_hfs(const char *path, size_t len)
+{
    • struct statfs buf;
    • char *p = ALLOCA_N(char, len+1);
    • memcpy(p, path, len);
    • p[len] = 0;
    • if (statfs(p, &buf) == 0) {
    • return buf.f_type == 17; /* HFS on darwin */
    • }
    • return FALSE;
    +}

+/*
    ◦ vpath is UTF8-MAC string
    • */
    +VALUE
    +compose_utf8_mac(VALUE vpath)
    +{
    • const char *path0 = RSTRING_PTR(vpath);
    • const char *p = path0;
    • const char *subpath = p;
    • const char *pend = RSTRING_END(vpath);
    • int hfs_p;
    • VALUE result;
    • rb_encoding *utf8, *utf8mac;
    • static VALUE utf8enc;

    • if (*p++ != '/')
    • return vpath;

    • if (!utf8enc) {
    • utf8mac = rb_enc_find("UTF8-MAC");
    • utf8 = rb_utf8_encoding();
    • utf8enc = rb_enc_from_encoding(utf8);
    • }

    • result = rb_str_buf_new(RSTRING_LEN(vpath));
    • hfs_p = is_hfs("/", 1);

    • for (; p < pend; p++) {
    • if (*p != '/')

    • ██████████

    • if (hfs_p != is_hfs(path0, p-path0)) {

    • ██████████

    • ████████████████████████████████████████████████████████████████████████████████

    • ████████████████████████████████████████████████████████████████████████████████

    • ████████████████████████████████████████████████████████████████████████████████

    • ██████████

    • ██████████

    • ████████████████████████████████████████████████████████████████████████████████

    • ██████████

    • ████████████████████████████████████████████████████████████████████████████████

    • ████████████████████████████████████████████████████████████████████████████████

    • }
    • }
    • if (hfs_p) {
    • VALUE str = rb_enc_str_new(subpath, p - subpath, utf8mac);
    • rb_str_buf_append(result, rb_str_encode(str, utf8enc, 0, Qnil));
```

```

• }
• else {
• rb_str_buf_cat(result, subpath, p - subpath);
• }

• rb_enc_associate(result, utf8);
• return result;
+}
+static VALUE
+dir_s_compose_path(VALUE dir, VALUE path)
+{
• return compose_utf8_mac(path);
+}
+endif

```

```

#define FNM_NOESCAPE 0x01
#define FNM_PATHNAME 0x02
#define FNM_DOTMATCH 0x04
@@ -2120,6 +2198,7 @@ Init_Dir(void)
rb_define_singleton_method(rb_cDir,"delete", dir_s_rmdir, 1);
rb_define_singleton_method(rb_cDir,"unlink", dir_s_rmdir, 1);
rb_define_singleton_method(rb_cDir,"home", dir_s_home, -1);

• rb_define_singleton_method(rb_cDir,"compose", dir_s_compose_path, 1);

rb_define_singleton_method(rb_cDir,"glob", dir_s_glob, -1);
rb_define_singleton_method(rb_cDir,"[]", dir_s_aref, -1);

```

#16 - 03/19/2013 05:06 PM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r39821.
Kenny, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

compose HFS file names

- dir.c (glob_helper): compose HFS file names from UTF8-MAC.
[\[ruby-core:48745\]](#) [Bug [#7267](#)]

#17 - 07/23/2014 10:11 AM - duerst (Martin Dürst)

- Related to Feature #10084: Add Unicode String Normalization to String class added

Files

test.rb	926 Bytes	11/02/2012	kennygrant (Kenny Grant)
Testé.txt	21 Bytes	11/02/2012	kennygrant (Kenny Grant)
results.txt	1.09 KB	11/02/2012	kennygrant (Kenny Grant)
writer.rb	221 Bytes	11/03/2012	kennygrant (Kenny Grant)