

## Re: not quite 1252

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2006-04/msg04226.html>

---

- *From:* Anton Vredegoor <[anton.vredegoor@xxxxxxxxxx](mailto:anton.vredegoor@xxxxxxxxxx)>
  - *Date:* Thu, 27 Apr 2006 17:27:24 +0200
- 

John Machin wrote:

Firstly, this should be 'content.xml', not 'contents.xml'.

Right, the code doesn't do *anything*\* :-( Thanks for pointing that out. At least it doesn't do much harm either :-|

Secondly, as pointed out by Sergei, the data is encoded by OOo as UTF-8 e.g. what is '\x94' in cp1252 is \u201d which is '\xe2\x80\x9d' in UTF-8. The kill\_gremlins function is intended to fix Unicode strings that have been obtained by decoding 8-bit strings using 'latin1' instead of 'cp1252'. When you pump '\xe2\x80\x9c' through the kill\_gremlins function, it changes the \x80 to a Euro symbol, and leaves the other two alone. Because the \x9d is not defined in cp1252, it then causes your code to die in a hole when you attempt to encode it as cp1252: UnicodeEncodeError: 'charmap' codec can't encode character u'\x9d' in position 1761: character maps to <undefined>

Yeah, converting to cp1252 was all that was necessary, like Sergei wrote.

I don't see how this code repairs anything (quite the contrary!), unless there's some side effect of just read/writestr. Enlightenment, please.

You're quite right. I'm extremely embarrassed now. What's left for me is just to explain how it got this bad.

First I noticed that by extracting from content.xml using OOo's getiterator function, some \x94 codes were left inside the document.

But that was an *artifact*\*, because if one prints something using s.\_\_repr\_\_() as is used for example when printing a list of strings (duh) the output is not the same as when one prints with 'print s'. I guess what is called then is str(s).

Ok, now we have that out of the way, I hope.

So I immediately posted a message about conversion errors, assuming something in the open office xml file

Re: not quite 1252

was not quite 1252. In fact it wasn't, it was UTF-8 like Sergei wrote, but it was easy to convert it to cp1252, no problem.

Then I also noticed that not all xml-tags were printed if I just iterated the xml-tree and filtered out only those elements with a text attribute, like 'if x.text: print x'

In fact there are a lot of printable things that haven't got a text attribute, for example some items with tag (xxxx)s.

When F pointed me to gremlins there was on this page the following text:

<quote>

Some applications add CP1252 (Windows, Western Europe) characters to documents marked up as ISO 8859-1 (Latin 1) or other encodings. These characters are not valid ISO-8859-1 characters, and may cause all sorts of problems in processing and display applications.

</quote>

I concluded that these \x94 codes (which I didn't know about them being a figment of my representation yet) were responsible for my iterator skipping over some text elements, but in fact the iterator skipped them because they had no text attribute even though they were somehow containing text.

Now add my natural tendency to see that what I think is the case rather than neutrally observing the world as it is into the mix and of course I saw the \x94 disappear (but that was because I now was printing them straight and not indirectly as elements of a list) and also I thought that now the xml-parsing 'errors' had disappeared but that was just because I saw some text element appear that I thought I hadn't seen before (but in fact it was there all the time).

One man's enlightenment sometimes is another's embarrassment, or so it seems. Thanks to you all clearing up my perceptions, and sorry about all the confusion I created.

What I want to know next is how to access and print the elements that contain text but have no text attribute, that is, if it's not to taxing on my badly damaged ego.

Anton