Can INI files be Unicode? Yes, they can, but it has to be your idea

devblogs.microsoft.com/oldnewthing/20240606-00

June 6, 2024

Raymond Chen

INI files were introduced by 16-bit Windows, and 16-bit Windows predates Unicode, so INI files naturally did not support Unicode at the time they were introduced. The relatively simple format of INI files means that many people parse (and sometimes even modify) them directly, without using the INI file manager. This in turn means that the format of INI files is pretty much locked and cannot be extended, since there is no mechanism for extending them in a way that won't break those manual INI file parsers.

This "locked in" nature of the INI file format means that even if you call the Unicode version, `WritePrivateProfileStringW`, the resulting INI file will not be Unicode. It will be a best approximation of your Unicode data in the ambient ANSI code page. The system doesn't know whether the INI file is going to be processed by somebody's homemade INI file parser, and writing it out in Unicode would break them.

You might think, "Aw, c'mon. If you use the Unicode `WritePrivateProfileStringW` function, then clearly the resulting INI file can be Unicode. After all, this is a new function, so there's no need to preserve legacy behavior." However, Michael Kaplan noted that this would mean that converting a program from ANSI to Unicode (which was a frequent occurrence back in the day) would invisibly modify file formats, and your program may not be ready for that.

But that doesn't mean that INI files could *never* support Unicode.

Because if the INI file was *already* Unicode, then there would be no harm in *keeping* it in Unicode. The decision to create a Unicode INI file came from somewhere else, and we're just following somebody else's decision.

So the rule is that the INI file functions will preserve Unicode-ness, but will never take it upon themselves to create a Unicode INI file. In particular, if you use a Write function to *create* an INI file, that INI file will be created as ANSI, for backward compatibility.

This behavior is called out in the documentation:

> If the file was created using Unicode characters, the function writes Unicode characters to the file. Otherwise, the function writes ANSI characters.

Michael says, “I have almost no idea what this text is trying to say, but I am 100% sure it is wrong.”

What it’s trying to say is what Michael inferred: The function writes Unicode characters to the file if the file is already Unicode.

I think what confused Michael was the phrase “If the file was created”. This is not referring to the creation of the file by the `WritePrivateProfileStringW` function itself, but rather to whether the file had *already been created* as a Unicode file before `WritePrivateProfileStringW` was called.

Arguably, the text could be made a little clearer:

> If the file already exists and consists of Unicode characters, the function writes Unicode characters to the file. Otherwise, the function writes ANSI characters.