File Name Support in Applications

Peter Occil

File Name Support in Applications

This version of the document is dated 2023-06-13.

Peter Occil

The issue of supporting file names is tricky, because different file systems—

- support different character encodings,
- use different rules in the storage of file names,
- use different rules for case comparisons and normalization comparisons,
- have different limits on the number of code units a file name can have, and
- could reject certain file names.

For example, by default—

- NTFS does case-sensitive comparisons of file names, but preserves the case of file names it stores;
- ext2 does a case-insensitive comparison of file names, and
- HFS Plus uses a particular normalization form in file name storage.

Most but not all modern file systems support file names with *non-basic code points* (names with code points beyond the Basic Latin range of the Unicode Standard). Such file names are called *internationalized file names* here.

In addition, certain file names are problematic; examples are "con", "aux", and other names reserved by earlier versions of the Windows operating system. (For Windows-specific information on file name support, see "**Naming Files, Paths, and Namespaces**" in Microsoft Docs.)

Applications that wish to support internationalized file names can follow the suggestions below.

0.1 Guidance for User-Facing Files

User-facing files are files created by end users or introduced into the application by end users. End users may want to name files in their language, making it necessary for many applications to support internationalized file names.

When creating new files: The MailLib library includes a `MakeFilename` method that converts a title or file name to a suitable name for saving data to

a file. `MakeFilename` does a number of things to maximize the chance that the name can be used as is in most file systems.

In one possible use of `MakeFilename`, a word-processing application could create a file name for a document by taking the document's title or the first few words of its body and adding a file extension like ".document" to those words (for example, "My Report.document"), then pass that name to the `MakeFilename` method to get a suggested file name to show a user seeking to save that document.

When accessing existing files: If an application receives the name of an existing file (as opposed to its directory path) from the file system, it should use that file name without change for the purposes of accessing or overwriting that file; this means that for such purposes, the application should treat that file name as uninterpreted data without converting its contents in any way, including by the `MakeFilename` method, a transcoder, or a case converter. This doesn't forbid applications from making changes to that file name for other purposes, including for the purpose of displaying that name to end users.

0.2 Guidance for Non-User-Facing Files

Internal files are files used by the application only and not exposed directly to end users.

To maximize compatibility with file system conventions, applications should limit the names of internal files to names that have the following characters only and are left unchanged by the **`MakeFilename`** method:

- Basic lower-case letters (U+0061 to U+007A).
- Basic digits (U+0030 to U+0039).
- Hyphen, full stop ("-", ".").
- Underscore ("_") if portability is not a concern (see RFC 2049 sec. 3).

In addition, such file names should not begin or end with "-" or "." or have two or more consecutive full stops (".."). (Basic upper-case letters, U+0041 to U+005a, are not suggested here because different file systems have different rules for case comparisons.)

Applications should avoid giving internal files an internationalized file name without a compelling reason to do so. This is especially because there are ways to encode such file names in this restricted character set, one of which is to—

- put the internationalized string **in UTF-8** (an 8-bit encoding form of the Unicode Standard), then
- encode the UTF-8 bytes using lowercase base16 or lowercase base32 without padding (RFC 4648).

Alternatively, applications could store internationalized or other names for an internal file separately from the file (such as in a "metadata file" or in a database table).

0.3 File Name Length Limits

Different file systems have different limits in the sizes of file names. To maximize compatibility with different file system limits, applications should avoid using file names longer than 63 Unicode code points.

(MS-DOS supported only file names with up to 8 bytes, followed optionally by "." and up to three more bytes, and with no more than one ".". Such a limit almost never occurs in practice today.)

0.4 Normalization and HFS Plus

The issue of normalization can come into play if an application supports internationalized file names.

The string returned by `MakeFilename` is normalized using Unicode normalization form C (NFC) (see the **PeterO.Text.NormalizerInput** class for details). Although most file systems preserve the normalization of file names, there is one notable exception: The HFS Plus file system (on macOS before High Sierra and on iOS before 10.3) stores file names using a modified version of normalization form D (NFD) in which certain code points are not decomposed, including all base + slash code points, which are the only composed code points in Unicode that are decomposed in NFD but not in HFS Plus's version of NFD. If the file name will be used to save a file to an HFS Plus storage device, it is enough to normalize the return value with NFD for this purpose (because all base + slash code points were converted beforehand by MakeFilename to an alternate form). See also Apple's Technical Q&A "Text Encodings in VFS" and Technical Note TN1150, "HFS Plus Volume Format".

0.5 Directory Names

The guidance given here applies to names of directories as they do to file names.