Internet Engineering Task Force
Internet-Draft
Intended status: Best Current Practice
Expires: January 7, 2021

N. Williams, Ed.
Cryptonector, LLC

July 6, 2020

Internationalization Considerations for Filesystems and Filesystem Protocols
draft-williams-filesystem-18n-00

Abstract

This document describes requirements for internationalization (I18N) of filesystems specifically in the context of Internet protocols, the architecture for filesystems in most currently popular general purpose operating systems, and their implications for filesystem I18N. From the I18N requirements for filesystems and the architecture of running code we derive requirements and recommendations for implementors of operating systems and/or filesystems, as well as for Internet remote filesystem protocols.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 7, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

[TBD: Add references galore. How to reference Unicode? How to reference US-ASCII? How best to reference HFS+? How best to reference ZFS? May have to find useful references for POSIX and WIN32. Various blog entries may be of interest -- can they be referenced?]

We, the Internet community, have long concluded that we must internationalize all our protocols. This is generally not an easy task, as often we are constrained by the realities of what can be achieved while maintaining backwards compatibility.

In this document we focus on filesystem internationalization (I18N), specifically only for file names and file paths. Here we address the two main issues that arise in filesystem I18N:

- Unicode equivalence

- Case foldings for case-insensitivity

These two issues are different flavors of the same generic issue: that there can be more than one way to write text with the same rendering and/or semantics.

Only I18N issues relating to file names and paths are addressed here. In particular, I18N issues related to representations of user identities and groups, for use in access control lists (ACLs) or other authorization systems, are out of scope for this document. Also out of scope here are I18N issues related to Uniform Resource Identifiers (URIs) [RFC3986] or Internationalized Resource Identifiers (IRIs) [RFC3987].

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Filesystem Internationalization

We must address two issues:

- Unicode equivalence
- Case foldings for case-insensitivity

Unicode can represent certain character strings in multiple visually- and semantically-equivalent ways. For example, there are two ways to express LATIN SMALL LETTER A WITH ACUTE (á):

- U+00E1
- U+0061 U+0301

For some glyphs there is a single way to write them. For others there are two. And for yet others there can be many more than two.

To deal with the equivalence problem, Unicode defines Normal Forms (NFs), of which there are two basic ones: Normal Form Composed (NFC), and Normal Form Decomposed (NFD). There are also NFs that use "compatibility" Foldings, NFKC and NFKD. Unicode-aware applications can normalize text to avoid ambiguities, or they can use form-insensitive string comparisons, or both.

Some filesystems support case-insensitivity, which is trivial to define and implement for US-ASCII, but non-trivial for Unicode,

requiring not only larger case-folding tables, but also localized case-folding tables as case-folding rules might differ from locale to locale.

1.2.1. Canonical Equivalence (Normalization)

For case-sensitive filesystems, only Unicode equivalence issues arise as to file names and file paths. These can be addressed in one of two ways:

- normalize file names when created and when looked up,
- perform form-insensitive string comparisons on lookup.

The first option yields normalized file names on-disk and on the wire (e.g., when listing directories). We shall term this "normalize-on-CREATE", or sometimes "normalize-on-CREATE-and-LOOKUP", or even just "NoCL".

The second option preserves form as originally produced by the user or on their behalf by their system's text input modes, but otherwise is form-insensitive. That is, this option permits either encoding of, e.g., LATIN SMALL LETTER A WITH ACUTE on-disk and on the wire, but permits only one form of any string, whether normal or not. We shall term this option "form-insensitive", or sometimes "form-insensitive and form-preserving", or just "FIP".

Unicode compatibility equivalence allows equivalence between different representations of the same abstract character that may nonetheless have different visual appearance of behavior. There are two canonical forms that support compatibility equivalence: NFKC and NFKD. Using NoCL with NFKC or NFKD may be surprising to users in a visual way. While form-insensitivity with NFKC or NFKD may surprise users who might consider two file names distinct even when Unicode considers them equivalent under compatibility equivalence. The latter seems less likely and less surprising, though that is an entirely subjective judgement.

We do not recommend either of NoCL or FIP over the other.

1.2.2. Case Foldings for Case-Insensitivity

Case-insensitivity implies folding characters of one case to another for comparison purposes, typically to lower-case. These case foldings are defined by Unicode. Generally, case-insensitive filesystems preserve original case just form-insensitive filesystems preserve original form.

It is possible that some case foldings may have to vary by locale. A commonly used example of character where case foldings that varies by locale is LATIN SMALL LETTER DOTLESS I (U+0131).

In some cases it may be possible to construct case-folding tailorings that are locale-neutral. For example, all of the following conuld be considered equivalent:

- LATIN CAPITAL LETTER I (U+0049)
- LATIN SMALL LETTER I (U+0069)
- LATIN CAPITAL LETTER I WITH DOT ABOVE (U+0130)
- LATIN SMALL LETTER DOTLESS I (U+0131)

which might satisfy a mix of users including those familiar with Turkish and those not, using the same filesystem.

1.2.3. Caching Clients

Remote filesystem protocols often involve caching on clients, which caching may require knowledge of filesystem I18N settings in order to permit local operations to be performed using cached directory listings that work the same way as on the server. We do not specify any case foldings here. Instead we will either create a registry of case folding tailorings, or use the Common Locale Data Repository (CLDR), then require that filesystems and servers be able to identify what case foldings are in effect for case-insensitive filesystems.

1.3. Running Code Architecture Notes

Surprisingly, almost all if not all general purpose operating systems in common use today have a "virtual filesystem switch" (VFS) [McKusick86] [wikipedia] [1] interface that permits the use of multiple different filesystem types on one system, all accessed through the same filesystems application programming interfaces (APIs). The VFS is essentially a pluggable layer that includes functionality for routing calls from user processes to the appropriate filesystems. The VFS has even been generalized and extended to support isolation, thus we have the Filesystem in Userspace (FUSE), which is akin to a remote filesystem protocol, but for use over local inter-process communications (IPC) facilities.

The VFS architecture was developed in the 1980s, before Unicode adoption. It is not surprising then that in general -if not simply always today- the code path from the interface between a user application and the operating system all the way to the filesystem

implements no I18N functionality whatsoever, and does the absolute minimum of character data interpretation:

- use of US-ASCII NUL (for "C string" termination),
- use of US-ASCII '/' and/or '\' (for file path component delimiting).

For example, the 4.4BSD operating system and derivatives have a VFS [BSD4.4], as do Solaris and derivatives [SolarisInternals], Windows <https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/>, OS X, and Linux. A VFS of a sort, including FUSE, may well be the only reasonable way to support more than one kind of filesystem while retaining compatibility with previously-existing filesystem APIs. This explains why so many modern operating systems have a VFS.

Thus in most if not all general purpose operating systems today, the code path from the boundary between the application and the operating system, and the boundary between the VFS and the filesystem, is "just-use-8" or "just-use-16" (as in UTF-16 [UNICODE]), with no attempt at normalization or case folding done anywhere in between.

There are filesystem servers that access raw storage directly and implement the filesystem and the remote filesystem protocol server in one monolythic stack without a VFS in the way, but it is very common to have remote filesystem protocol servers implemented on top of the VFS or on top of the system calls. Even monolythic servers tend to support a notion of multiple filesystems in a server or volume, and may have different I18N settings for each filesystem. Thus it's common to leave I18N handling to code layers close to the filesystem even in monolythic server implementations.

In practice all of foregoing has led to I18N functionality residing strictly in the filesystem. Two filesystems have defined the best current practices in this regard:

- HFS+, which does normalize-on-CREATE (and LOOKUP), normalizing to a form that is very close to NFD and is case-sensitive;
- ZFS, which implements form-insensitive, form-preserving behavior and optionally implements case-insensitive, case-preserving behavior on a per-filesystem basis.

Altogether, these circumstances make it very difficult to reliably and always locate I18N functionality above the VFS, or to not use a VFS at all: there are too many places to alter, and all must agree exactly on I18N choices. Moreover, implementing case-insensitive but case-preserving behavior above the VFS requires fully reading each

directory, and so does implementing form-insensitive and form-preserving behavior at the VFS layer itself. The only behaviors that can be reliably implemented at or above the VFS are normalize- and case-fold-on-CREATE (and LOOKUP).

Consider the set of already-running code that must all be modified in order to reliably implement I18N above the filesystem on general purpose operating systems:

- filesystem protocol servers, including but not limited to:
  - Network File System (NFSv4) [RFC7530];
  - Hypertext Transfer Protocol (HTTP) servers serving resources hosted on filesystems[RFC7230];
  - SSH File Transfer Protocol (SFTP) [I-D.ietf-secsh-filexfer];
  - various remote filesystem protocols that are not Internet Protocols (i.e., not standards-track Internet RFCs);
- POSIX system call layers or user process system call stub libraries;
- WIN32 system call layers or user process system call stub libraries.

Regarding system calls and system call stubs in user process system libraries, the continued use of statically-linked executables means that these cannot reliably be modified. Indeed, on some systems the Application Binary Interface (ABI) between user-space applications and the operating system kernel is well-defined and long-term stable. The system call handlers cannot reliably inspect the calling process to determine any attributes of its locale. Adding new system calls is possible, but existing running code wouldn't use them. For similar reasons, the VFS layer is generally (always) completely unaware of any attributes of the locale of applications calling it, whether via system calls or any other path.

Unix-like operating systems are generally (always) "just-use-8", assuming only that file names and paths are C strings (i.e., terminated by zero-valued bytes) and sufficiently compatible with US-ASCII that the file path component separator character, US-ASCII '/', is meaningful. As a result, it is possible to find I18N-unaware filesystems with one or more non-Unicode, non-ASCII codesets in use for file names! We leave non-ASCII and non-Unicode file names out of scope here.

For these reasons it is simply not practical to implement I18N at any layer above the VFS.

Even in the VFS, form- and case-insensitive and -preserving behaviors would be difficult to implement as performantly as in the filesystem. The VFS would have to list a directory completely before being able to apply those behaviors. It is reasonable to expect caching clients of remote filesystems to cache directory listings (especially for offline operation), but it isn't reasonable to expect the same of the VFS. Compare to the filesystem itself, which can maintain a fast index (e.g., hash table or b-tree) where the keys are normalized and possibly case-folded file names and thus may not need to read directories in order to perform fast lookups that are form- and even case-insensitive.

The only way to implement I18N behaviors in the VFS layer rather than at the filesystem is to abandon form- and case-preserving behaviors. For case-insensitivity this would require using sentence-case, or all lower-case, perhaps, and all such choices would surely be surprising to users. At any rate, that approach would also render much running code "non-compliant" with any Internet filesystem protocol I18N specification.

Therefore, generally speaking, only the filesystem can reliably, interoperably, and performantly implement I18N behaviors in general purpose operating systems.

Note that variations in I18N behaviors can happen even on the same server with multiple filesystems of the same type. This can happen because of

- different Unicode versions being used at the times of creation of various filesystems, and

- different locale settings on various filesystems.

Locale variations are only relevant to case-folding for case-insensitivity. Running code mostly uses default case-folding rules, but there is no reason to assume that locale-specific case-folding rules won't be supported by running code in the future.

It may not be possible or easy for a filesystem to adopt new Unicode versions, or adopt backwards-incompatible case foldings, after content has been created in it that would be ambiguous under new rules. This implies that where a client for a remote filesystem must know what I18N functionality to implement for use with cached directory listings, the client must know specifically what profile of I18N functionality each cached filesystem implements.

2. Filesystem I18N Guidelines

We begin be recognizing and accepting that much running code implements I18N functionality at the filesystem. Given this, we catalogue the range of acceptable behaviors. Filesystems adhering to this specification MUST implement only acceptable I18N behaviors as specified here. Acceptable variations may be registered in a to-be-determined (IANA?) registry of filesystem I18N behaviors.

2.1. Filesystem I18N Guidelines: Non-Unicode File names

- Filesystems SHOULD reject attempts to create new non-Unicode file names.
- Filesystems either MUST normalize on CREATE (and LOOKUP), or MUST be form-insensitive and form-preserving.
- Filesystems MUST specify a Unicode version for their equivalence behaviors.

2.2. Filesystem I18N Guidelines: Case-Insensitivity

- Filesystems MAY support case-insensitivity, in which case they SHOULD be case-preserving. Filesystems that are case-insensitive but not case-preserving either MUST specify a case form, such as title case or sentence case.
- Case foldings for case-insensitive filesystems MUST be identified. The Unicode default case foldings SHOULD be the default case algorithms for the identified Unicode version without additional tailorings. Filesystems that use case algorithms tailored to specific locales SHOULD use case foldings registered in a to-be-determined (IANA?) registry.
- Case-insensitive filesystems MUST specify a Unicode version for their case-insensitive behavior.

2.3. I18N Versioning

Each filesystem MUST identify a Unicode version for their I18N behaviors. Filesystem implementations SHOULD adopt new Unicode versions as they are produced, though it is understood that it may be difficult to migrate non-empty filesystems to new Unicode versions.

3. Filesystem Protocol I18N Guidelines

Remote filesystem protocols that allow clients to perform lookups against cached directory listings MUST allow clients to discover all relevant I18N behaviors of the filesystem whence any given directory listing:

- whether the filesystem normalizes on CREATE (and LOOKUP), and if so, to what NF in what Unicode version;
- whether the filesystem is form-insensitive and form-preserving, and if so, in what Unicode version;
- whether the filesystem is case-insensitive and case-preserving, and if so, with what foldings (default or tailured, and if tailored provide an identifier for the set of foldings), and a Unicode version.

Foldings are identified via a folding set name as registered in a to-be-determined (IANA?) registry.

Because some filesystems might allow for different I18N settings on a per-directory basis, remote filesystem protocols MUST allow those settings to be discoverable on a per-directory basis.

Internet filesystem servers MUST reject attempts to create new non-Unicode file names. (Note that this requirement is weaker ("SHOULD") for the actual filesystems, since those might have to allow non-Unicode content for legacy reasons via interfaces other than Internet filesystem protocols.)

3.1. I18N and Caching in Filesystem Protocol Clients

Caching clients of remote filesystems either MUST NOT perform lookups against cached directory listings, or MUST query the directories' filesystems' I18N profiles and apply the same I18N equivalent form policis and case-insensitivity case foldings.

4. Internationalization Considerations

This document deals in internationalization throughout.

5. IANA Considerations

[ALTERNATIVELY use locale names and CLDR? Need to determine the stability of CLDR locales... Basically, we need stable locale names, and stable case-folding mappings.]

We hereby request the creation of a new IANA registry with Expert Review registration rules with the following fields:

- name, an identifier-like name
- Unicode version number
- listing of case folding tailorings and/or references to external case folding tailoring specifications

The case foldings registered here will be used by case-insensitive filesystems and filesystem protocols to identify tailored case foldings so that caching clients can implement the same case-insensitive behavior using cached directory listings.

6. Security Considerations

Security considerations of Unicode and filesystem protocols apply. No new security considerations are added or need be noted here.

The methods of handling equivalent Unicode strings cause aliasing. This is not expected to be a security problem.

Case-insensitivity causes aliasing. This is not expected to be a security problem.

No effort is made here to handle confusables. This is not expected to be a serious security problem in the context of file servers.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 12.1.0", May 2019, <https://www.unicode.org/versions/Unicode12.1.0/>.

7.2. Informative References

[BSD4.4] McKusik, M., Bostic, K., Karels, M., and J. Quarterman, "The Design and Implementation of the 4.4BSD Operating System", DOI 10.5555/231070, 1996.

[I-D.ietf-secsh-filexfer] Galbraith, J. and O. Saarenmaa, "SSH File Transfer Protocol", draft-ietf-secsh-filexfer-13 (work in progress), July 2006.

[McKusick86] McKusik, M. and M. Karels, "Towards a Compatible File System Interface", Jun 1986.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7530] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 Protocol", RFC 7530, DOI 10.17487/RFC7530, March 2015, <https://www.rfc-editor.org/info/rfc7530>.

[SolarisInternals] McDougal, R. and J. Mauro, "Solaris Internals -- Solaris 10 and OpenSolaris Kernel Architecture", 2007.

7.3. URIs

[1] https://en.wikipedia.org/wiki/Virtual_file_system

Author's Address

Nico Williams (editor)
Cryptonector, LLC
Austin, TX
USA

Email: nico@cryptonector.com