NFSv4
Internet-Draft
Updates: 5661, 7530 (if approved)
Intended status: Standards Track
Expires: September 10, 2020

D. Noveck
NetApp

March 9, 2020

Internationalization for the NFSv4 Protocols draft-dnoveck-nfsv4-internationalization-01

Abstract

This document describes the handling of internationalization for all NFSv4 protocols, including NFSv4.0, NFSv4.1, NFSv4.2 and extensions thereof, and future minor versions.

It updates RFC7530 and RFC5661.

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 September 10, 2020.

Copyright Notice

Table of Contents

1. Introduction

Internationalization is a complex topic with its own set of terminology (see [19]). The topic is made more complex for the NFSv4 protocols by the tangled history described in Section 3. This document is based on the actual behavior of NFSv4 client and server implementations (for all existing minor versions) and is intended to serve as a basis for further implementations to be developed that can interact with existing implementations as well as those to be developed in the future.

Note that the behaviors on which this document are based are each demonstrated by a combination of an NFSv4 server implementation proper and a server-side physical file system. It is common for servers and physical file systems to be configurable as to the behavior shown. In the discussion below, each configuration that shows different behavior is considered separately.

As a consequence of this choice, normative terms defined in RFC2119 [1] are derived from implementation behavior, rather than the other way around, as is more commonly the case. The specifics are discussed in Section 2.

With regard to the question of interoperability with existing specifications for NFSv4 minor versions, different minor versions pose different issues.

- With regard to NFSv4.0 as defined in RFC7530 [3], no significant interoperability issues are expected to arise because the internationalization in that specification, which is the basis for this one, was also based on the behavior of existing implementations. Although, in a formal sense, the treatment of internationalization here supersedes that in RFC7530 [3], the treatments are intended to be essentailly the same in order to eliminate interoperability issues.

  Because of a change in the handling of Internationalized domain names, there are some differences from the handling in RFC7530 [3], as discussed in Section 3. For a discussion of those differences and potential compatibility issues, see Sections 8.1 and 8.2.

- With regard to NFSv4.1 as defined RFC5661 [4], the situation is quite different. The approach to internationalization specified in that document was never implemented, and implementers were either unaware of the troublesome implications of that approach or chose to ignore the existing specification as essentially unimplementable. An internationalization approach compatible with that specified in RFC7530 [3] tended to be followed, despite the fact that, in other respects, NFSv4.1 considered to be a separate protocol.

  If there were NFSv4 servers who obeyed the internationalization dictates within RFC5661 [4], or clients that expected servers to do so, they would fail to interoperate with typical clients and servers when dealing with non-UTF8 file names, which are quite common. As no such implementation have come to our attention, it has to be assumed that they do not exist and interoperability with existing implementations as described here is an appropriate basis for this document.

2. Requirements Language

2.1. Requirements Language Definition

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 BCP 14 [1] [2] when, and only when, they appear in all capitals, as shown here.

2.2. Requirements Language Derivation

Although the key words "MUST", "SHOULD", and "MAY" retain their normal meanings, as described above, we need to explain how the statements involving these terms were defined. In the case of statements within Section 8, these derive from the requirements of other internet specifications. However, in other cases, this specification's descriptions was derived from existing implementation patterns requiring that we explain how the normative terms used derive from the behavior of existing implementations, in those situations in which existing implementation behavior patterns can be determined.

- Behavior implemented by all existing clients or servers is described using "MUST", since new implementations need to follow existing ones to be assured of interoperability. While it is possible that different behavior might be workable, we have found no case where this seems reasonable.

  The converse holds for "MUST NOT": if a type of behavior poses interoperability problems, it MUST NOT be implemented by any existing clients or servers.

- Behavior implemented by most existing clients or servers, where that behavior is more desirable than any alternative, is described using "SHOULD", since new implementations need to follow that existing practice unless there are strong reasons to do otherwise.

  The converse holds for "SHOULD NOT".

- Behavior implemented by some, but not all, existing clients or servers is described using "MAY", indicating that new implementations have a choice as to whether they will behave in that way. Thus, new implementations will have the same flexibility that existing ones do.

- Behavior implemented by all existing clients or servers, so far as is known -- but where there remains some uncertainty as to details -- is described using "should". Such cases primarily concern details of error returns. New implementations should follow

existing practice even though such situations generally do not affect interoperability.

There are also cases in which certain server behaviors, while not known to exist, cannot be reliably determined not to exist. In part, this is a consequence of the long period of time that has elapsed since the publication of the defining specifications, resulting in a situation in which those involved in t implementation work may no longer be involved in or aware of working group activities.

In the case of possible server behavior that is neither known to exist nor known not to exist, we use "SHOULD NOT" and "MUST NOT" as follows, and similarly for "SHOULD" and "MUST".

- In some cases, the potential behavior is not known to exist but is of such a nature that, if it were in fact implemented, interoperability difficulties would be expected and reported, giving us cause to conclude that the potential behavior is not implemented. For such behavior, we use "MUST NOT". Similarly, we use "MUST" to apply to the contrary behavior.

- In other cases, potential behavior is not known to exist but the behavior, while undesirable, is not of such a nature that we are able to draw any conclusions about its potential existence. In such cases, we use "SHOULD NOT". Similarly, we use "SHOULD" to apply to the contrary behavior.

In the case of a "MAY", "SHOULD", or "SHOULD NOT" that applies to servers, clients need to be aware that there are servers that may or may not take the specified action, and they need to be prepared for either eventuality.

3. History

The history of internationalization within NFSv4 is discussed in this section. Despite the fact that NFSv4.0 and subsequent minor versions have differed in many ways, the actual implementations of internationalization have remained the same and internationalized names have been handled without regard to the minor version being used. As a result, this document is able to treat internationalization for all NFSv4 minor versions together.

During the period from the publication of RFC3010 [14] until now, two different perspectives with regard to internationalization have been held and represented, to varying degrees, in specifications for NFSv4 minor versions.

- The perspective held by NFSv4 implementers treated most aspects of internationalization as basically outside the scope of what NFSv4 client and server implementers could deal with. This was because the POSIX interface treated filenames as uninterpreted strings of bytes, because the file systems used by NFSv4 servers treated filenames similarly, and because those file systems contained files with internationalized names using a number of different encoding methods, chosen by the users of the POSIX interface. From this perspective, wider support for internationalized names and general use of universal encodings was a matter for users and applications and not for protocol implementers or designers.

- Within the IETF in general and in the IESG, there was a feeling that new protocols, such as NFSv4, could not avoid dealing with internationalization issues, making it difficult to treat these matters, as the implementers' perspective would have it, as essentially out of scope.

As specifications were developed, approved, and at times rewritten, this fundamental difference of approach was never fully resolved, although, with the publication of RFC7530 [3], a satisfactory modus vivendi may have been arrived at.

Although many specifications were published dealing with NFSv4 internationalization, all minor versions used the same implementation approach, even when the current specification for that minor version specified an entirely different approach. As a result, we need to treat the history of NFSv4 internationalization below as an integrated whole, rather than treating individual minor versions separately.

- The approach to internationalization specified in RFC3010 [14] sidestepped the conflict of approaches cited above by discussing the reasons that UTF-8 encoding was desirable while leaving filenames as uninterpreted strings of bytes. The issue of string normalization was avoided by saying "The NFS version 4 protocol does not mandate the use of a particular normalization form at this time."

  Despite this approach's inconsistency with general IETF expectations regarding internationalization, RFC3010 was published as a Proposed Standard. NFSv4.0 implementation related to internationalization of filenames followed the same paradigm used by NFSv3, assuring interoperability with files created using that protocol, as well as with those created using local means of file creation.

- When it became necessary, because of issues with byte-range locking, to create an rfc3010bis, no change to the previously approved approach seemed indicated and the drafts submitted up until [21] closely followed RFC3010 as regards internationalization. The IESG then decided that a different approach to internationalization was required, to be based on stringprep [15] and rfc3010bis was accordingly revised, replacing all of the Internationalization section, before being published as RFC3530 [18].

  These changes required the rejection of file names that were not valid UTF-8, file names that included code points not, at the time of publication, assigned a Unicode character (e.g. capital eszett) or that were not allowed by stringprep (e.g. Zero-width joiner and non-joiner characters). Because these restrictions would have caused the set of valid file names to be different on NFS-mounted and local file systems there was no chance of them ever being implemented.

  Because these specification changes were made without working group involvement, most implementers were unaware of them while those who were aware of the changes ignored them and continued to develop implementations based on the internationalization approach specified in RFC3010.

- When NFsv4.1 was being developed, it seemed that no changes in internationalization would be required. Many people were unaware of the stringprep-based requirements which made the NFSv4.0 internationalization specified in RFC3530 unimplementable. As a result, the internationalization specified in RFC5661 [4] was the same as that in RFC3530.

  As a result, even though NFSv4.1 was a separate protocol and could have had a different approach to internationalization, for a considerable time, internationalization for both protocols was specified to be the same (in RFC3530 and RFC5661) while the actual implementations of the two minor versions both followed the approach specified in RFC3010, despite its obsoleted status.

- When work started on rfc3530bis it was clear that issues related to internationalization had to be addressed. When the implications of the stringprep references in RFC3530 were discussed with implementers it became clear that mandating that NFSv4.0 filenames conform to stringprep was not appropriate. While some working group members articulated the view that, because of the need to maintain compatibility with the POSIX interface and existing file systems, internationalization for NFSv4 could not be successfully addressed by the IETF, the

rfc3530bis draft submitted to the IESG did not explicitly embrace the implementers' perspective set forth above.

The draft submitted to the IESG and RFC7530 [3] as published provided an explanation (see Section 4) as to why restrictions on character encodings were not viable. It allowed non-UTF-8 encodings to be used for internationalized filenames while defining UTF-8 as the preferred encoding and allowing servers to reject non-UTF-8 string as invalid. Other stringprep-based string restrictions were eliminated. With regard to normalization, it continued to defer the matter, leaving open the possibility that one might be chosen later.

This approach is compatible, in implementation terms, with that specified in RFC3010 [14], allowing it to be used compatibly with existing implementations for all existing minor versions. This is despite the fact that RFC5661 [4] specifies an entirely different approach.

As a result of discussions leading up to the publishing of RFC7530, it was discovered that some local file systems used with NFSv4 were configured to be both normalization-aware and normalization-preserving, mapping all canonically equivalent file names to the same file while preserving the form actually used to create the file, of whatever form, normalized or not. This behavior, which is legal according to RFC3010, which says little about name mapping is probably illegal according to stringprep. Nevertheless, it was expressly pointed out in RFC7530 as a valid choice to deal with normalization issues, since it allows normalization-aware processing without the difficulties that arise in imposing a particular normalization form, as described in Section 7.

In its discussion of internationalized domain names, RFC7530 [3] adopted an approach compatible with IDNA2003, rather than attempting to derive the specification from the behavior of existing implementations.

- When IDNA2003 was replaced by IDNA2008, the internationalization specified by [3] was not changed. Also, it appears unlikely that implementations were changed to reflect that shift.

- NFSv4.2 made no changes to internationalization. As a result, RFC7862 [5] which made no mention of internationalization, implicitly aligned internationalization in NFSv4.2 with that in NFSv4.1, as specified by RFC5661 [4].

As a result of this implicit alignment, there is no need for this document to specifically address NFSv4.2 or be marked as updating RFC7862. It is sufficient that it updates RFC5661, which specifies the internationalization for NFSv4.1, inherited by NFSv4.2.

The above history, can, for the purposes of the rest of this document be summarized in the following statements:

- The actual treatment of internationalization within NFSv4 has not been affected by the particular minor version used, despite the fact that the specifications for the minor versions have often differed in their treatment of internationalization.

- With regard to filenames, implementations have followed the internationalization approach specified in RFC3010, which is compatible with the treatment in RFC7530.

- With regard to internationalized domain names, RFC7530 [3] specified an approach compatible with IDNA at the time of publication. However, no detailed analysis was done to determine whether NFSv4 implementations actually followed that approach

In order to deal with all NFSv4 minor versions, this document follows the internationalization approach defined in RFC7530, with some changes motivated by the shift from IDNA2003 to IDNA2008. The intention is to maintain compatibility with all existing NFSv4 minor versions. Potential compatibility issues with regard to the IDNA shift are discussed in Section 8.2. Issues relating to potential future minor versions and protocol extensions are dealt with in Section 11.

4. Limitations on Internationalization-Related Processing in the NFSv4 Context

There are a number of noteworthy circumstances that limit the degree to which internationalization-related encoding and normalization-related restrictions can be made universal with regard to NFSv4 clients and servers:

- The NFSv4 client is part of an extensive set of client-side software components whose design and internal interfaces are not within the IETF's purview, limiting the degree to which a particular character encoding might be made standard.

- Server-side handling of file component names is typically implemented within a server-side physical file system, whose

handling of character encoding and normalization is not specifiable by the IETF.

- Typical implementation patterns in UNIX systems result in the NFSv4 client having no knowledge of the character encoding being used, which might even vary between processes on the same client system.

- Users may need access to files stored previously with non-UTF-8 encodings, or with UTF-8 encodings that are not in accord with any particular normalization form.

5. Summary of Server Behavior Types

Servers MAY reject component name strings that are not valid UTF-8. This leads to a number of types of valid server behavior, as outlined below. When these are combined with the valid normalization-related behaviors as described in Section 6, this leads to the combined behaviors outlined below.

- Servers that limit file component names within a given file system to UTF-8 strings exist with normalization-related handling as described in Section 6. These are best described as behaving as "UTF-8-only servers".

- Servers that do not limit file component names on particular file systems to UTF-8 strings are very common and are necessary to deal with clients/applications not oriented to the use of UTF-8. Such servers ignore normalization-related issues, and there is no way for them to implement either normalization or representation-independent lookups. These are best described as behaving as "UTF-8-unaware servers" for such file systems, since they treat file component names as uninterpreted strings of bytes and have no knowledge of the characters represented. See Section 9 for details.

- It is possible for a server to allow component names that are not valid UTF-8, while still being aware of the structure of UTF-8 strings. Such servers could implement either normalization or representation-independent lookups but apply those techniques only to valid UTF-8 strings. Such servers are not common, but it is possible to configure at least one known server to have this behavior. This behavior SHOULD NOT be used due to the possibility that a filename using one encoding may, by coincidence, have the appearance of a UTF-8 filename; the results of UTF-8 normalization or representation-independent lookups are unlikely to be correct in all cases, when considered from the viewpoint of the other encoding.

6. String Encoding

Strings that potentially contain characters outside the ASCII range [10] are generally represented in NFSv4 using the UTF-8 encoding [8] of Unicode [11]. See [8] for precise encoding and decoding rules.

Some details of the protocol treatment depend on the type of string:

- For strings that are component names, the preferred encoding for any non-ASCII characters is the UTF-8 representation of Unicode.

  In many cases, clients have no knowledge of the encoding being used, with the encoding done at the user level under the control of a per-process locale specification. As a result, it may be impossible for the NFSv4 client to enforce the use of UTF-8. The use of non-UTF-8 encodings can be problematic, since it may interfere with access to files stored using other forms of name encoding. Also, normalization-related processing (see Section 7) of a string not encoded in UTF-8 could result in inappropriate name modification or aliasing. In cases in which one has a non-UTF-8 encoded name that accidentally conforms to UTF-8 rules, substitution of canonically equivalent strings can change the non-UTF-8 encoded name drastically.

  The kinds of modification and aliasing mentioned here can lead to both false negatives and false positives, depending on the strings in question, which can result in security issues such as elevation of privilege and denial of service (see [20] for further discussion).

- For strings based on domain names, non-ASCII characters MUST be represented using the UTF-8 encoding of Unicode, and additional string format restrictions may apply. See Section 8 for details.

- The contents of symbolic links (of type linktext4 in the XDR) MUST be treated as opaque data by NFSv4 servers. Although UTF-8 encoding is often used, it need not be. In this respect, the contents of symbolic links are like the contents of regular files in that their encoding is not within the scope of this specification.

- For other sorts of strings, any non-ASCII characters SHOULD be represented using the UTF-8 encoding of Unicode.

7. Normalization

The client and server operating environments can potentially differ in their policies and operational methods with respect to character normalization (see [11] for a discussion of normalization forms). This difference may also exist between applications on the same client. This adds to the difficulty of providing a single normalization policy for the protocol that allows for maximal interoperability. This issue is similar to the issues of character case where the server may or may not support case-insensitive filename matching and may or may not preserve the character case when storing filenames. The protocol does not mandate a particular behavior but allows for a range of useful behaviors.

The NFSv4 protocol does not mandate the use of a particular normalization form. A subsequent minor version of the NFSv4 protocol might specify a particular normalization form, although there would be difficulties in doing so (see Section 11 for details). In any case, the server and client can expect that they may receive unnormalized characters within protocol requests and responses. If the operating environment requires normalization, then the implementation will need to normalize the various UTF-8 encoded strings within the protocol before presenting the information to an application (at the client) or local file system (at the server).

Server implementations MAY normalize filenames to conform to a particular normalization form before using the resulting string when looking up or creating a file. Servers MAY also perform normalization-insensitive string comparisons without modifying the names to match a particular normalization form. Except in cases in which component names are excluded from normalization-related handling because they are not valid UTF-8 strings, a server MUST make the same choice (as to whether to normalize or not, the target form of normalization, and whether to do normalization-insensitive string comparisons) in the same way for all accesses to a particular file system. Servers SHOULD NOT reject a filename because it does not conform to a particular normalization form, as this would deny access to clients that use a different normalization form or clients acting on behalf of application that use a different normalization form.

8. String Types with Processing Defined by Other Internet Areas

There are two types of strings that NFSv4 deals with that are based on domain names. Processing of such strings is defined by other Internet standards, and hence the processing behavior for such strings should be consistent across all server operating systems and server file systems.

This section differs from other sections of this document in two respects:

- The normative statements within this section are not derived from the behavior from existing NFSv4 implementations, but derive instead from existing RFCs.

- Because of the switch from IDNA2003 [16] [17] to IDNA2008 [6], this section is necessarily different from the corresponding section (i.e. Section 12.6) of [3]. The differences are discussed in Section 8.1.

Because of this shift, there could be compatibility issues to be expected between implementations obeying Section 12.6 of [3] and those following this document. Whether such compatibility issues actually exist depends on the behavior of NFSv4 implementations and how domain names are actually used in existing implementations. These matters will be discussed in Section 8.2.

The types of strings referred to above are as follows:

- Server names as they appear in the fs_locations and fs_locations_info attribute. Notes that for most purposes, such server names will only be sent by the server to the client. The exception is the use of these attributes in a VERIFY or NVERIFY operation.

- Principal suffixes that are used to denote sets of users and groups, and are in the form of domain names.

The general rules for handling all of these domain-related strings are similar and independent of the role of the sender or receiver as client or server, although the consequences of failure to obey these rules may be different for client or server. The server can report errors when it is sent invalid strings, whereas the client will simply ignore an invalid string or use a default value in its place.

The string sent SHOULD be in the form of one or more unvalidated U-labels as defined by [6]. In cases where this cannot be done, the string will instead be in the form of one or more LDH labels [6]. The receiver needs to be able to accept domain and server names in any of the formats allowed. The server MUST reject, using the error NFS4ERR_INVAL, any of the following:

- a string that is not valid UTF-8.

- a string that contains an XN-label (begins with "xn--") for which the characters after "xn--" are not valid output of the Punycode algorithm [7].

- a string that contains a reserved LDH label which is not an XN-label.

When a domain string is part of id@domain or group@domain, there are two possible approaches:

1. The server generally treats the domain string as a series of unvalidated U-labels. In cases where the domain string is a series of unvalidated A-labels or Non-Reserved LDH (NR-LDH) labels, it converts them to U-labels using the Punycode algorithm [7]. As a result, the domain string returned within a user id on a GETATTR may not match that sent when the user id is set using SETATTR, although when this happens, the domain will be in the form of an unvalidated U-label.

2. The server treats the domain string as a series of unvalidated U-labels. Specifically, it does not map a domain string that is not a U-label into a U-label using the methods described above. As a result, the domain string returned on a GETATTR of the user id MUST be the same as that used when setting the user id by the SETATTR.

A server SHOULD use the first method.

For VERIFY and NVERIFY, additional string processing requirements apply to verification of the owner and owner_group attributes; see the section entitled "Interpreting owner and owner_group" for the document specifying the minor version in question (RFC750 [3], RFC5661 [4])

8.1. Effect of IDNA Changes

Overall, the effect of the shift to IDNA2008 is to limit the degree of understanding of the IDNA-based restrictions on domain names that were expected of NFSv4 in RFC7530 [3]. Despite this specification, the degree to which implementations actually implemented such restrictions is open to question and will be discussed in detail in Section 8.2

In analyzing how various cases are to be dealt with according to RFC7530, there a number of troubling uncertainties that arise in trying to interpret the existing specification:

- There are a number of cases in which "SHOULD" is used that are confusing. According to RFC2119 [1], "SHOULD" means that "there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course". To fully understand a particular "SHOULD", there needs to be enough context to determine whether particular reasons for ignoring the item are in fact valid, and sufficient guidance to understand the implication of ignoring the item. In the absence of such information, the relevant fact is that the peer needs to deal with the item being ignored, making the implications of a "SHOULD" hard to distinguish from those of "MAY".

- While the document states. "the general rules for handling all of these domain-related strings are similar and independent of the role of the sender or receiver as client or server", all of the following text is explicitly about the server's options, choices and responsibilities, leaving the client case unclear.

- In a number of places within the paragraph describing server approach #1, the word "can" is used as in the text "the server can use the ToUnicode function", leaving it unclear whether the server can choose to do anything else and if so what.

The following cases are those where RFC7530 requires use of IDNA handling and this requirement could, if implementations follow them, create potential compatibility issues, which need to be understood.

- The degree to which RFC3490 [16] requires that characters other than U+002E (full stop) be treated as label separators, including U+3002 (ideographic full stop), U+FF0E (fullwidth full stop), U+FF61 (halfwidth ideographic full stop).

- The degree to which RFC3490 [16] that server or client needs to validate a putative A-label or U-label or to rectify it if it is not valid.

8.2. Potential Compatibility Issues Related to IDNA Changes

There are a number of factors relating to the handling of domain names within NFSv4 implementations that are important in understanding why any compatibility issues might be less troubling than a comparison of the two IDNA approaches might suggest:

- Much of the potentially conflicting IDNA-related behavior required or recommended for the server by RFC7530 [3] might not actually be implemented, limiting the potential harmful effects of ceasing to mandate it.

- Even if such behavior were implemented by servers, no compatibility issue would arise unless clients actually relied on the server to implement it. Given that none of this behavior is made required, the chances of that occurring is quite small.

- The range of potential values for user and group attributes sent by clients are often quite small with implementations commonly restricting all such values to a single domain string. This is even though RFCs 7530 [3] and 5661 [4] are written without mention of such restrictions.

  Specification of users and groups in the "id@domain" format within NFSv4 was adopted to enable expansion of the spaces of users and groups beyond the 32-bit id spaces mandated in NFSv3 [13] and NFsv2 [12]. While one obstacle to expansion was eliminated, most implementations were unable to actually effect that expansion, principally because the physical file systems used assume that user and group identifiers fit in 32 bits each and the vnode interfaces used by server implementations make similar assumptions.

  Given these restrictions, the typical implementation pattern is for servers to accept only a single domain, specified as part of the server configuration, together with information necessary to effect the appropriate name-to-id mappings.

- The other uses of domain names in NFSv4, to represent hostnames in location attributes, the values are generated by the server and will normally include only include hostnames within DNS-registered domains.

Keeping the above in mind, we can see that interoperability issues, while they might exist are unlikely to raise major challenges as looking to the following specific cases shows

- When an internationalized domain name is used as part of a user or group, it would need to be configured as such, with the domain string known to both client and server.

  While it is theoretically possible that a client might work with an invalid domain string and rely on the server to correct it to an IDNA-acceptable one, such a scenario has to be considered extremely unlikely, since it would depend on multiple servers implementing the same correction, especially since there is no evidence of such corrections ever having been implemented by NFSv4 servers.

- When an internationalized domain in a location string is meant to specify a registered domain, similar considerations apply.

  While it is theoretically possible that a client might work with an invalid domain string and rely on the server to correct it to the appropriate registered one, such a scenario has to be considered extremely unlikely, since it would depend on multiple servers implementing the same correction, especially since there is no evidence of such corrections ever having beeen inimplemented by NFSv4 servers.

- When an internationalized domain in a location string is meant to specify a non-registered domain, any such server-applied corrections would be useless.

  In this situation, any potential interoperability issue would arise from rejecting the name, which has to be considered as what should have been done in the first place.

9. Errors Related to UTF-8

Where the client sends an invalid UTF-8 string, the server MAY return an NFS4ERR_INVAL error. This includes cases in which inappropriate prefixes are detected and where the count includes trailing bytes that do not constitute a full Multiple-Octet Coded Universal Character Set (UCS) character.

Requirements for server handling of component names that are not valid UTF-8, when a server does not return NFS4ERR_INVAL in response to receiving them, are described in Section 10.

Where the string supplied by the client is not rejected with NFS4ERR_INVAL but contains characters that are not supported by the server as a value for that string (e.g., names containing slashes, or characters that do not fit into 16 bits when converted from UTF-8 to a Unicode codepoint), the server should return an NFS4ERR_BADCHAR error.

Where a UTF-8 string is used as a filename, and the file system, while supporting all of the characters within the name, does not allow that particular name to be used, the server should return the error NFS4ERR_BADNAME. This includes such situations as file system prohibitions of "." and ".." as filenames for certain operations, and similar constraints.

10. Servers That Accept File Component Names That Are Not Valid UTF-8 Strings

As stated previously, servers MAY accept, on all or on some subset of the physical file systems exported, component names that are not valid UTF-8 strings. A typical pattern is for a server to use UTF-8-unaware physical file systems that treat component names as uninterpreted strings of bytes, rather than having any awareness of the character set being used.

Such servers SHOULD NOT change the stored representation of component names from those received on the wire and SHOULD use an octet-by-octet comparison of component name strings to determine equivalence (as opposed to any broader notion of string comparison). This is because the server has no knowledge of the character encoding being used.

Nonetheless, when such a server uses a broader notion of string equivalence than what is recommended in the preceding paragraph, the following considerations apply:

- Outside of 7-bit ASCII, string processing that changes string contents is usually specific to a character set and hence is generally unsafe when the character set is unknown. This processing could change the filename in an unexpected fashion, rendering the file inaccessible to the application or client that created or renamed the file and to others expecting the original filename. Hence, such processing should not be performed, because doing so is likely to result in incorrect string modification or aliasing.

- Unicode normalization is particularly dangerous, as such processing assumes that the string is UTF-8. When that assumption is false because a different character set was used to create the filename, normalization may corrupt the filename with respect to that character set, rendering the file inaccessible to the application that created it and others expecting the original filename. Hence, Unicode normalization SHOULD NOT be performed, because it may cause incorrect string modification or aliasing.

When the above recommendations are not followed, the resulting string modification and aliasing can lead to both false negatives and false positives, depending on the strings in question, which can result in security issues such as elevation of privilege and denial of service (see [20] for further discussion).

11. Future Minor Versions and Extensions

As stated above, all current NFSv4 minor versions allow use of non-UTF-8 encodings, allow servers a choice of whether to be aware of normalization issues or not, and allows servers a number of choices about how to address normalization issues. This range of choices reflects the need to accommodate existing file systems and user expectations about character handling which in turn reflect the assumptions of the POSIX model of handling file names.

While it is theoretically possible for a subsequent minor version to change these aspects of the protocol (see [9]), this section will explain why any such change is highly unlikely, making it expected that these aspects of NFSv4 internationalization handling will be retained indefinitely. As a result, any new minor version specification document that made such a change would have to be marked as updating or obsoleting this document

No such change could be done as an extension to an existing minor version or in a new minor version consisting only of OPTIONAL features. Such a change could only be done in a new minor version, which like minor version one, was prepared to be incompatible to some degree with the previous minor versions. While it appears unlikely that such minor versions will be adopted, the possibility cannot be excluded, so we need to explore the difficulties of changing the aspects of internationalization handling mentioned above.

- Establishing UTF-8 as the sole means of encoding for internationalized characters, would make inaccessible existing files stored with other encodings. Further, unless there were a corresponding change in the UNIX file interface model, it would cause the set of valid names for local and remote files to diverge.

- Imposing a particular normalization form, in the sense of refusing to create to allow access to files whose UTF-8-encoded names are not of the selected normalization form would give rise to similar difficulties.

- Defining a preferred normalization form to be returned as the names of all internationalized files, would result in applications having to deal with sudden unexplained changes of file names for existing files.

None of the above appears likely since there does not seem to be any corresponding benefits to justify the difficulties that they would create.

There would also be difficulties in otherwise reducing the set of three acceptable normalization handling options, without reducing it to a single option by imposing a specific normalization form.

- Eliminating the possibility of a single possible normalization form, would pose similar difficulties to imposing the other one, even if representation-independent comparisons were also allowed.

  In either case, a specific normalization form would be disfavored, with no corresponding benefit.

- Allowing only representation-independent lookups would not impose difficulties for clients, but there are reasons to doubt it could be universally implemented, since such name comparisons would have to be done within the file system itself.

  Such a change could only be made once support file system support for representation-independent file lookups would become commonly available. As long as the POSIX file naming model continues its sway, that would be unlikely to happen.

One possible internationalization-related extension that the working could adopt would be definition of an OPTIONAL per-fs attribute defining the internationalization-related handling for that file system. That would allow clients to be aware of server choices in this area and could be adopted without disrupting existing clients and servers.

12. IANA Considerations

The current document does not require any actions by IANA.

13. Security Considerations

Unicode in the form of UTF-8 is generally is used for file component names (i.e., both directory and file components). However, other character sets may also be allowed for these names. For the owner and owner_group attributes and other sorts strings whose form is affected by standard outside NFSv4 (see Section 8.) are always encoded as UTF-8. String processing (e.g., Unicode normalization) raises security concerns for string comparison. See Sections 8 and 7 as well as the respective Sections 5.9 of RFC7530 [3] and RFC5661 [4] for further discussion. See [20] for related identifier comparison security considerations. File component names are identifiers with respect to the identifier comparison discussion in [20] because they are used to identify the objects to which ACLs are applied (See the respective Sections 6 of RFC7530 [3] and RFC5661 [4]).

14. References

14.1. Normative References

[1] 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>.

[2] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[3] 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>.

[4] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, DOI 10.17487/RFC5661, January 2010, <https://www.rfc-editor.org/info/rfc5661>.

[5] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 Protocol", RFC 7862, DOI 10.17487/RFC7862, November 2016, <https://www.rfc-editor.org/info/rfc7862>.

[6] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[7] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, DOI 10.17487/RFC3492, March 2003, <https://www.rfc-editor.org/info/rfc3492>.

[8] 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>.

[9] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", RFC 8178, DOI 10.17487/RFC8178, July 2017, <https://www.rfc-editor.org/info/rfc8178>.

[10] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, October 1969, <http://www.rfc-editor.org/info/rfc20>.

[11] The Unicode Consortium, "The Unicode Standard, Version 7.0.0", (Mountain View, CA: The Unicode Consortium, 2014 ISBN 978-1-936213-09-2), June 2014, <http://www.unicode.org/versions/latest/>.

14.2. Informative References

[12] Nowicki, B., "NFS: Network File System Protocol specification", RFC 1094, DOI 10.17487/RFC1094, March 1989, <https://www.rfc-editor.org/info/rfc1094>.

[13] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, DOI 10.17487/RFC1813, June 1995, <https://www.rfc-editor.org/info/rfc1813>.

[14] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "NFS version 4 Protocol", RFC 3010, DOI 10.17487/RFC3010, December 2000, <https://www.rfc-editor.org/info/rfc3010>.

[15] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, DOI 10.17487/RFC3454, December 2002, <https://www.rfc-editor.org/info/rfc3454>.

[16] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, DOI 10.17487/RFC3490, March 2003, <https://www.rfc-editor.org/info/rfc3490>.

[17] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, DOI 10.17487/RFC3491, March 2003, <https://www.rfc-editor.org/info/rfc3491>.

[18] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, DOI 10.17487/RFC3530, April 2003, <https://www.rfc-editor.org/info/rfc3530>.

[19] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <https://www.rfc-editor.org/info/rfc6365>.

[20] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <https://www.rfc-editor.org/info/rfc6943>.

[21] Shepler, S., "NFS version 4 Protocol", draft-ietf-nfsv4-rfc3010bis-04 (work in progress), October 2002.

Appendix A. Acknowledgements

This document is based, in large part, on Section 12 of [3] and all the people who contributed to that work, have helped make this document possible, including David Black, Peter Staubach, Nico Williams, Mike Eisler, Trond Myklebust, James Lentini, Mike Kupfer and Peter Saint-Andre.

The author wishes to thank Tom Haynes for his timely suggestion to pursue the task of dealing with internationalization on an NFSv4-wide basis.

Author's Address

David Noveck
NetApp
1601 Trapelo Road
Waltham, MA 02451
United States of America

Phone: +1 781 572 8038
Email: davenoveck@gmail.com