Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

A. Sullivan
Dyn
A. Freytag
ASMUS Inc.

March 9, 2015

A Problem Statement to Motivate Work on Locale-free Unicode Identifiers

draft-sullivan-lucid-prob-stmt-00

Abstract

Internationalization techniques that the IETF has adopted depended on some assumptions about the way characters get added to Unicode. Some of those assumptions turn out not to have been true. Discussion is necessary to determine how the IETF should respond to the new understanding of how Unicode works.

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 http://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, 2015.

Copyright Notice

Table of Contents

1. Introduction

Among its features, IDNA2008 [RFC5890] [RFC5891] [RFC5892] [RFC5893] [RFC5894] [RFC5895] provides a way of using Unicode [Unicode] characters without regard to the version of Unicode available. The same approach is generalized for protocols other than DNS by the PRECIS framework [I-D.ietf-precis-framework].

The mechanism used is called "inclusion", and is outlined in Section 2.1 below. We call the general strategy "inclusion-based identifier internationalization" or "i3" for short. I3 depends on certain assumptions made in the IETF at the time it was being developed. Some of those assumptions were about the relationships between various characters and the likelihood that similar such relationships would get added to future versions of Unicode. Those

assumptions turn out not to have been true in every case. This raises a question, therefore, about whether the current approach meets the needs of the IETF for internationalizing identifiers.

This memo attempts to give enough background about the situation so that IETF participants can participate in a discussion about what (if anything) to do about the state of affairs; the discussion is expected to happen as part of the LUCID BoF at IETF 92. The reader is assumed to be familiar with the terminology in [RFC6365]. This memo owes a great deal to the exposition in [I-D.klensin-idna-5892upd-unicode70].

2. Background

The intent of Unicode is to encode all known writing systems into a single coded character set. One consequence of that goal is that Unicode encodes an enormous number of characters. Another is that the work of Unicode does not end until every writing system is encoded; even after that, it needs to continue to track any changes in those writing systems. Unicode encodes abstract characters, not glyphs. Because of the way Unicode was built up over time, there are sometimes multiple ways to encode the same abstract character. If Unicode encodes an abstract character in more than one way, then for most purposes the different encodings should all be treated as though they're the same character. This is called "canonical equivalence".

A lack of a defined canonical equivalence is tantamount to an assertion by Unicode that the two encodings do not represent the same abstract character, even if both happen to result in the same appearance.

Every encoded character in Unicode (that is, every code point) is associated with a set of properties. The properties define what script a code point is in, whether it is a letter or a number or punctuation and so forth, what direction it is written in, to what other code point or code point sequence it is canonically equivalent, and many other properties. These properties are important to the inclusion mechanism.

2.1. The Inclusion Mechanism

Because of both the enormous number of characters in Unicode and the many purposes it must serve, Unicode contains characters that are not well-suited for use as part of identifiers for network protocols. The inclusion mechanism starts by assuming an empty set of characters. It then evaluates Unicode characters not individually, but instead by classifying them according to their properties. This

classification provides the "derived properties" that IDNA2008 and PRECIS rely upon.

In practice, the inclusion mechanism includes code points that are letters or digits. There are some ways to include or exclude characters that otherwise would be excluded or included (respectively); but it is impractical to evaluate each character, so most characters are included or excluded based on the properties they have.

I3 depends on the assumption that strings that will be used in identifiers will not have any ambiguous matching to other strings. In practice, this means that input strings to the protocol are expected to be in Normalization Form C. This way, any alternative sequences of code points for the same characters will be normalized to a single form. Assuming then that those characters are all included by the inclusion mechanism, the string is eligible to be an identifier under the protocol.

2.2. The Difference Between Theory and Practice

In principle, under i3 identifiers should be unambiguous. It has always been recognized, however, that for humans some ambiguity was inevitable, because of the vagaries of writing systems and of human perception.

Normalization Form NFC removes the ambiguities based on dual or multiple encoding for the same abstract character. However, characters are not the same as their glyphs. This means that it is possible for certain abstract characters to share a glyph. We call such abstract characters "homoglyphs". While this looks at first like something that should be handled (or should have been handled) by normalization (NFC or something else), there are important differences; the situation is in some sense an extreme case of a spectrum of ambiguity discussed in the following section.

2.2.1. Confusability

While Unicode deals in abstract characters and i3 works on Unicode code points, users interact with the characters as actually rendered: glyphs. There are characters that, depending on font, sometimes look quite similar to one another (such as "l" and "1"); any character that is like this is often called "visually similar". More difficult are characters that, in any normal rendering, always look the same as one another. The shared history of Cyrillic, Greek, and Latin scripts, for example, means that there are characters in each script that function similarly and that are usually indistinguishable from one another, though they are not the same abstract character. These

are examples of "homoglyphs." Any character that can be confused for another one can be called confusable, and confusability can be thought of as a spectrum with "visually similar" at one end, and "homoglyphs" at the other. (We use the term "homoglyph" strictly: code points that normally use the same glyph when rendered.)

Most of the time, there is some characteristic that can help to mitigate confusion. Mitigation may be as simple as using a font designed to distinguish among different characters. For homoglyphs, a large number of cases (but not all of them) turn out to be in different scripts. As a result, there is an operational convention that identifiers should always be in a single script. (This strategy can be less than successful in cases where each identifier is in a single script, but the repertoire used in operation allows multiple scripts, because of whole string confusables -- strings made up entirely of homoglyphs of another string in a different script.)

There is another convention that operators should only ever use the smallest repertoire of code points possible for their environment. So, for example, if there is a code point that is sometimes used but is perhaps a little obscure, it is better to leave it out and gain some experience with other cases first. In particular, code points used in a language with which the administrator is not familiar should probably be excluded. In the case of IDNA, some client programs restrict display of U-labels to top-level domains known to have policies about single-script labels. None of these policies or convention will do anything to help strict homoglyphs of each other in the same script (see Appendix A for some example cases.)

2.2.1.1. Not everything can be solved

Before continuing, it is worth noting that there are some cases that, regardless of mitigation, are fundamentally impossible to solve. There are certainly cases of two strings in which all the code points in one script in the first string, and all the code points in another script in the second string, are respectively confusable with one another. In that case, the strings cannot be distinguished by a reader, and the whole string is confusable. Further, human perception is easily tricked, so that entirely unrelated character sequences can become confusable, for example "rn" being confused with "m".

Given the facts of history and the contingencies of writing systems, one cannot defend against all of these cases; and it seems all but certain that many of these cases cannot successfully be addressed on the protocol level alone. In general, the i3 strategy can only define rules for one identifier at a time, and has no way to offer guidance about how different identifiers under the same scheme ought

to interact. Humans are likely to respond according to the entire identifier string, so there seems to be a deep tension between the narrow focus of i3, and the actual experience of users.

In addition, several factors limit the ability to ensure that any solution adopted is final and complete: the sheer complexity of writing systems, the fact that many of them are not equally well understood as Latin or Han, and that many less developed writing systems are potentially susceptible to paradigm changes as digital support for them becomes more widespread. Detailed knowledge about, and implementation experience for, these writing systems only emerges over time; disruptive changes are neither predictable ahead of time nor preventable. In essence, any solution to eliminate ambiguity can be expected to get some detail wrong.

Nobody should imagine that the present discussion takes as its goal the complete elimination of all possible confusion. The failure to achieve such a goal does not mean, however, that we should do nothing, any more than the low chances of ever arresting all grifters means that we should not enact laws against fraud. Our discussion, then, must focus on those problems that are able to be addressed in the constraint of the protocols; and, in particular, the subset that are suitable for that

2.2.2. The Problem Now Before Us

During the expert review necessary for supporting Unicode 7.0.0 for use with IDNA, a new code point U+08A1, ARABIC LETTER BEH WITH HAMZA ABOVE came in for some scrutiny. Using versions of Unicode up to and including 7.0.0, it is possible to combine ARABIC LETTER BEH (U+0628) and ARABIC HAMZA ABOVE (U+0654) to produce a glyph that is indistinguishable from the one produced by U+08A1. But U+08A1 and \u'0628'\u'0654' are not canonically equivalent. (For more discussion of this issue, see [I-D.klensin-idna-5892upd-unicode70].)

Further investigation reveals that there are several similar cases. ARABIC HAMZA ABOVE (U+0654) turns out to be implicated in some cases, but not all of them. There are cases in Latin (see Appendix A for examples). There are certainly cases in other scripts (some examples are provided in Appendix A). The majority of cases all have a handful of things in common:

- There are at least two forms by which the same glyph is produced.
- One of the forms uses a combining sequence and another form is a precomposed character, or else one of the forms is a digraph. [[CREF1: Is this true? Are there any cases that don't match it? --ajs]]

- The results when rendered as glyphs cannot be distinguished from one another.

- The two forms are not canonically equivalent.

- All of the relevant code points have the same script property, or else inherit the script property of the previous character so that it is not possible to select on the basis of the script.

- Competent users of the writing system in a language do not treat one of the combining sequence or the precomposed character as reasonable. To writers for whom the combining sequence is "wrong", it is not a case of a base character modified by an additional mark, but instead a separate letter. Conversely, to writers for whom the precomposed character is "wrong", it is definitely a matter of adding something to a character that otherwise stands on its own. (Not every possible combination would normally be used by anyone, of course, and sometimes -- not infrequently -- one of the alternatives is not used by any orthography.)

Cases that match these conditions might be considered to involve "non-normalizable diacritics", because most of the combining marks in question are non-spacing marks that are or act like diacritics.

3. Identifiers

Part of the reason i3 works from the assumption that not all Unicode code points are appropriate for identifiers is that identifiers do not work like words of phrases in a language. First, identifiers often appear in contexts where there is no way to tell the language of the identifiers. Indeed, many identifiers are not really "in a language" at all. Second, and partly because of that lack of linguistic root, identifiers are often either not words or use unusual orthography precisely to differentiate themselves.

In ordinary language use, the ambiguity identified in Section 2.2 may well create no difficulty. Running text has two properties that make this so. First, because there is a linguistic context (the rest of the text), it is possible to detect code points that are used in an unusual way and flag them or, even, create automatic rules to "fix" such issues. Second, linguistic context comes with spelling rules that automatically determine whether something is written the right way. Because of these facts, it is often possible even without a locale identifier to work out what the locale of the text ought to be. So, even in cases where passages of text need to be compared, it is possible to mitigate the issue.

The same locale-detection approach does not work for identifiers. Worse, identifiers, by their very nature, are things that must provide reliable exact matches. The whole point of an identifier is that it provides a reliable way of uniquely naming the thing to be identified. Partial matches and heuristics are inadequate for those purposes. Identifiers are often used as part of the security practices for a protocol, and therefore ambiguity in matching presents a risk for the security of any protocol relying on the identifier.

3.1. Types of Identifiers

It is worth observing that not all identifiers are of the same type. There are four relevant dimensions in which identifiers can differ in type:

1. Scope
    (a) Internet-wide
    (b) Unique within a context (often a site)
    (c) Link-local only
2. Management
    (a) Centrally managed
    (b) Contextually managed (e.g. registering a nickname with a server for a session)
    (c) Unmanaged
3. Durability
    (a) Permanent
    (b) Durable but with possible expiration
    (c) Temporary
    (d) Ephemeral
4. Authority
    (a) Single authority
    (b) Multiple authorities (possibly within a hierarchy)

(c) No authority

These different dimensions present ways in which mitigation of the identified issue might be possible. For instance, a protocol that uses only link-local identifiers that are unmanaged, temporary, and configured automatically does not really present a problem, because for practical purposes its linguistic context is constrained to the social realities of the LAN in question. A durable Internet-wide identifier centrally managed by multiple authorities will present a greater issue unless locale information comes along with the identifier.

4. Possible Nature of Problem

We may regard this problem as one of several different kinds, and depending on how we view it we will have different approaches to addressing it.

4.1. Just a Species of Confusables

Under this interpretation, the current issue is no different to any other confusable case, except in detail. Since there is no way to solve the general problem of confusables, there is no way to solve this problem either. Moreover, to the degree that confusables are solved outside protocols, by administration and policy, the current issue might be addressed by the same strategy.

This interpretation seems unsatisfying, because there exist some partial mitigations, and if suitable further mitigations are possible it would be wise to apply them.

4.2. Just a Species of Homoglyphs

Under this interpretation, the current issue is no different than any other homoglyph case. After all, the basic problem is that there is no way for a user to tell which codepoint is represented by what the user sees in either case.

There is some merit to this view, but it has the problem that many of the homoglyph issues (admittedly not all of them) can be mitigated through registration rules, and those rules can be established without examining the particular code points in question (that is, they can operate just on the properties of code points, such as script membership). The current issue does not allow such mitigation given the properties that are currently available. At the same time, it may be that it is impossible to deal with this adequately, and some judgement will be needed for what is adequate. This is an area where more discussion is clearly needed.

4.3. Separate Problem

Under this interpretation, there is a definable problem, and its boundaries can be specified.

That we can list some necessary conditions for the problem suggests that it is a separable problem. The list of factors in Section 2.2.2 seems to indicate that it is possible to describe the bounds of a problem that can be addressed separately.

What is not clear is whether it is separable enough to make it worth treating separately.

4.4. Unimportant Problem

Under this interpretation, while it is possible to describe the problem, it is not a problem worth addressing since nobody would ever create such identifiers on purpose.

The problem with this approach, for identifiers, is that it represents an opportunity for phishing and other similar attacks. While mitigation will not stop all such attacks, we should try to understand opportunities for those attacks and close when we have identified them and it is practical to do so.

Whether phishing or other attacks using confusable code points "pay off" depends to some extent on the popularity or frequency of the code points in question. While it may be worth to address the generalized issue, individual edge cases may have no practical consequences. The inability to address them then, should not hold up progress on a solution for the more common, general case.

5. Possible Ways Forward

There are a few ways that this issue could be mitigated. Note that this section is closely related to Section 3 in [I-D.klensin-idna-5892upd-unicode70].

5.1. Find the Cases, Disallow New Ones, and Deal With Old Ones

In this case, it is necessary to enumerate all the cases, add exceptions to DISALLOW any new cases from happening, and make a determination about what to do for every past case. There are two reasons to doubt whether this approach will work.

1. The IETF did not catch these issues during previous internationalization efforts, and it seems unlikely that in the

meantime it has acquired enough expertise in writing systems to do a proper job of it this time.

2. This approach blunts the effectiveness of being Unicode version-agnostic, since it would effectively block any future additions to Unicode that had any interaction with the present version.

So, this approach does not seem too promising.

5.2. Disallow Certain Combining Sequences Absolutely

In this case, instead of treating all the code points in Unicode, the IETF would need only to look at all combining characters. While the IETF obviously does not have the requisite expertise in writing systems to do this unilaterally, the Unicode Consortium does. In fact the Unicode Technical Committee has a clear understanding that some combining sequences are never intended to be used for orthographic purposes. Any glyph needed for an orthography or writing system will, once identified, be added as a single code point with "pre-composed" glyph.

In principle there is no obstacle, in these cases, to asking Unicode to express this understanding in form of a character property, which then means that IETF could DISALLOW the combining marks having such a property.

5.3. Do Nothing, Possibly Warn

One possibility is to accept that there is nothing one can do in general here, and that therefore the best one can do is warn people to be careful.

The problem with this approach, of course, is that it all but guarantees future problems with ambiguous identifiers. It would provide a good reason to reject all internationalized identifiers as representing a significant security risk, and would therefore mean that internationalized identifiers would become "second class". Unfortunately, however, the demand for internationalized identifiers would not likely be reduced by this decision, so some people would end up using identifiers with known security problems.

This approach may be the only possible in some of the borderline cases where mitigation approaches are not successful.

5.4. Identify Enough Commonality for a New Property

There is reason to suppose that, if the IETF can come up with clear and complete conditions under which code points causing an issue could be classified, the Unicode Technical Committee would add such a property to code points in future versions of the Unicode Standard. Assuming the conditions were clear, future additions to the Standard could also be assigned appropriate values of the property, meaning that the IETF could revert to making decisions about code points based on derived properties. Beyond the property mentioned in Section 5.2 this property could cover certain combining marks in the Arabic script.

If this is possible, it seems a desirable course of action.

5.5. Create an IETF-only Normalization Form

Under this approach, the IETF creates a special normalization form that it maintains outside the Unicode Standard. For the sake of the discussion, we'll call this "NFI".

This option does not seem workable. The IETF would have to evaluate every new release of Unicode to discover the extent to which the new release interacts with NFI. Because it would be independently maintained, Unicode stability guarantees would not apply to NFI; the results would be unpredictable. As a result, either the IETF would have to ignore new additions to Unicode, or else it would need UTC to take NFI into account. If UTC were able to do so, this option reduces to the option in Section 5.4. The UTC might not be able to do this, however, because the very principles that Unicode uses to assign new characters in certain situations guarantees that new characters will be added that cannot be so normalized and yet are essential for still-to-be-encoded writing systems. Communities for which these new characters would be added would also not accept any existing code point sequence as equivalent. This also means that Unicode cannot create a stability policy to take into account the needs of such an NFI.

6. Acknowledgements

The discussion in this memo owes a great deal to the IAB Internationalization program, and particularly to John Klensin.

7. Informative References

[I-D.ietf-precis-framework]
Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", draft-ietf-precis-framework-23 (work in progress), February 2015.

[I-D.klensin-idna-5892upd-unicode70]
Klensin, J. and P. Faeltstroem, "IDNA Update for Unicode 7.0.0", draft-klensin-idna-5892upd-unicode70-03 (work in progress), January 2015.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.

[RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.

[RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.

[RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.

[Unicode] "The Unicode Standard", http://www.unicode.org/versions/Unicode7.0.0/, .

Appendix A. Examples

There are a number of cases that illustrate the combining sequence or digraph issue:

U+08A1 vs \u'0628'\u'0654' This case is ARABIC LETTER BEH WITH HAMZA ABOVE, which is the one that was detected during expert review that caused the IETF to notice the issue. The issue existed before this, but we did not know it. For detailed discussion of this case and some of the following ones, see [I-D.klensin-idna-5892upd-unicode70]

U+0681 vs \u'062D'\u'0654' This case is ARABIC LETTER HAH WITH HAMZA ABOVE, which (like U+08A1) does not have a canonical equivalent. In both cases, the places where hamza above are used are specialized enough that the combining marks can be excluded in some cases (for example, the root zone under IDNA).

U+0623 vs \u'0627'\u'0654' This case is ARABIC LETTER ALEF WITH HAMZA ABOVE. Unlike the previous two cases, it does have a canonical equivalence with the combining sequence. In the past, the IETF misunderstood the reasons for the difference between this pair and the previous two cases.

U+09E1 vs u\'098C'u\'09E2' This case is BENGALI LETTER VOCALIC LL. This is an example in Bengali script of a case without a canonical equivalence to the combining sequence. Per Unicode, the single code point should be used to represent vowel letters in text, and the sequence of code points should not be used. But it is not a simple matter of disallowing the combining vowel mark in cases like this; where the combination does not exist and the use of the sequence is already established, Unicode is unlikely to encode the combination.

U+019A vs \u'006C'\u'0335' This case is LATIN SMALL LETTER L WITH BAR. In at least some fonts, there is a detectable difference with the combining sequence, but only if one types them one after another and compares them. There is no canonical equivalence here. Unicode has a principle of encoding barred letters as composites when needed for any writing system.

U+00F8 vs \u'006F'\u'0337' This is LATIN SMALL LETTER O WITH STROKE. The effect are similar to the previous case. Unicode has a principle of encoding stroked letters as composites when needed for any writing system.

U+02A6 vs \u'0074'\u'0073' This is LATIN SMALL LETTER TS DIGRAPH, which is not canonically equivalent to the letters t and s. The intent appears to be that the digraph shows the two shapes as kerned, but the difference may be slight out of context.

U+01C9 vs \u'006C'\u'006A' Unlike the TS digraph, the LJ digraph has a relevant compatibility decomposition, so it fails the relevant

stability rules under i3 and is therefore DISALLOWED. This illustrates the way that consistencies that might be natural to some users of a script are not necessarily found in it, possibly because of uses by another writing system.

U+06C8 vs u\'0648'u\'0670' ARABIC LETTER YU is an example where the normally-rendered character looks just like a combining sequence, but are named differently. In other words, this is an example where the simple fact of the Unicode name would have concealed the apparent relationship from the casual observer.

U+069 vs \u'0069'\u'0307' LATIN SMALL LETTER I followed by COMBINING DOT ABOVE by definition, renders exactly the same as LATIN SMALL LETTER I by itself and does so in practice for any good font. The same would be true if "i" was replaced with any of the other Soft_Dotted characters defined in Unicode. The character sequence \u'0069'\u'0307' (followed by no other combining mark) is reportedly rather common on the Internet. Because base character and stand-alone code point are the same in this case, and the code points affected have the Soft_Dotted property already, this could be mitigated separately via a context rule affecting U+0307.

Other cases test the claim that the issue lies primarily with combining sequences at all:

U+0B95 vs U+0BE7 The TAMIL LETTER KA and TAMIL DIGIT ONE are always indistinguishable, but needed to be encoded separately because one is a letter and the other is a digit.

Arabic-Indic Digits vs. Extended Arabic-Indic Digits Seven digits of these two sequences have entirely identical shapes. This case is an example of something dealt with in i3 that nevertheless can lead to confusions that are not fully mitigated. IDNA, for example, contains context rules restricting the digits to one set or another; but such rules apply only to a single label, not to an entire name. Moreover, it provides no way of distinguishing between two labels that both conform to the context rule, but where each contains one of the seven identical shapes.

U+53E3 vs U+56D7 These are two Han characters (roughly rectangular) that are different when laid side by side; but they may be impossible to distinguish out of context or in small print.

Authors' Addresses

Andrew Sullivan
Dyn
150 Dow St.
Manchester, NH 03101
US

Email: asullivan@dyn.com

Asmus Freytag
ASMUS Inc.

Email: asmus@unicode.org