Network Working Group
Internet-Draft
Updates: 5892, 5894 (if approved)
Intended status: Standards Track
Expires: April 11, 2018
October 8, 2017

J. Klensin

P. Faltstrom
Netnod

IDNA Update for Unicode 7.0 and Later Versions
draft-klensin-idna-5892upd-unicode70-05

Abstract

The current version of the IDNA specifications anticipated that each new version of Unicode would be reviewed to verify that no changes had been introduced that required adjustments to the set of rules and, in particular, whether new exceptions or backward compatibility adjustments were needed. The review for Unicode 7.0.0 first identified a potentially problematic new code point and then a much more general and difficult issue with Unicode normalization. This specification discusses those issues and proposes updates to IDNA and, potentially, the way the IETF handles comparison of identifiers more generally, especially when there is no associated language or language identification. It also applies an editorial clarification to RFC 5892 that was the subject of an earlier erratum and updates RFC 5894 to point to the issues involved.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Note in/about -04 and -05 Drafts: These two versions of the document contains a very large amount of new material as compared to the -03 version. The new material reflects an evolution of community understanding in the first quarter of 2015 and further evolution between then and mid-2017 from an assumption that the problem involved only a few code points and one combining character in a single script (Hamza Above and Arabic) to an understanding that the problem we have come to call "non-decomposing code points" and several closely related ones are quite pervasive and may represent fundamental misunderstandings or omissions from IDNA2008 (and, by extension, the basics of PRECIS [RFC8264]) that must be corrected if those protocols are going to be used in a way that supports internationalized identifiers on the Internet predictably (as seen by the end user) and securely.

This version is still necessarily incomplete: not only is our understanding probably still not comprehensive, but there are a number of placeholders for text and references. Nonetheless, the document in its current form should be useful as both the beginning of a comprehensive overview is the issues and a source of references to other relevant materials.

This draft could almost certainly be better organized to improve its readability: specific suggestions would be welcome.

1.1. Origins and Discovery of the Issue

The current version of the IDNA specifications, known as "IDNA2008" [RFC5890], anticipated that each new version of Unicode would be reviewed to verify that no changes had been introduced that required adjustments to IDNA’s rules and, in particular, whether new exceptions or backward compatibility adjustments were needed. When that review was carefully conducted for Unicode 7.0.0 [Unicode7], comparing it to prior versions including the text in Unicode 6.2 [Unicode62], it identified a problematic new code point (U+08A1, ARABIC LETTER BEH WITH HAMZA ABOVE). The code point was added for Arabic Script use with the Fula (also known as Fulfulde, Pulaar, amd Pular’Fulaare) language. That language is apparently most often written in Latin characters today [Omniglot-Fula] [Dalby] [Daniels].

The specific problem is discussed in detail in Section 3. In very broad terms, IDNA (and other IETF work) assume that, if one can represent "the same character" either as a combining sequence or as a single code point, strings that are identical except for those alternate forms will compare equal after normalization. Part of the difficulty that has characterized this discussion is that "the same" differs depending on the criteria that are chosen. It may be further complicated in practice by differences in preferred type styles or rendering, but Unicode code point choices are not supposed to depend on type style (font) variations and, again, IDNA has no mechanism for specifying language choices that might affect rendering.

The behavior of the newly-added code point, while non-optimal for IDNA, follows that of a few code points that predate Unicode 7.x and even the IDNA 2008 specifications and Unicode 6.0. Those existing code points, which may not be easy to accurately characterize as a group, make the question of what, if anything, to do about this new exceedingly problematic one and, perhaps separately, what to do about existing sets of code points with the same behavior, because different reasonable criteria yield different decisions, specifically:

- To disallow it (and future, but not existing, characters with similar characteristics) as an IDNA exception case creates inconsistencies with how those earlier code points were handled.

- To disallow it and the similar code points as well would necessitate invalidating some potential labels that would have been valid under IDNA2008 until this time. Depending on how the collection of similar code points is characterized, a few of them are almost certainly used in reasonable labels.

- To permit the new code point to be treated as PVALID creates a situation in which it is possible, within the same script, to compose the same character symbol (glyph or grapheme) in two different ways that do not compare equal even after normalization. That condition would then apply to it and the earlier code points with the same behavior. That situation contradicts a fundamental assumption of IDNA that is discussed in more detail below.

NOTE IN DRAFT:

This working draft discusses six alternatives, including an idea (an IETF-specific normalization form) that seemed too drastic to be considered when IDNA2008 was designed or even when the review of Unicode 7.0 for IDAN purposes began. In retrospect, it not only would have been appropriate to discuss when the IDNA2008 specifications were being developed but is appearing more attractive now. The authors suggest that the community discuss the relevant tradeoffs and make a decision and that the document then be revised to reflect that decision, with the other alternatives discussed as options not chosen. Because there is no ideal choice, the discussion of the issues in Section 3 is probably as or more important than the particular choice of how to handle this code point. In addition to providing information for this document, that section should be considered as an updating addendum to RFC 5894 [RFC5894] and should be incorporated into any future revision of that document.

As the result of this version of the document containing several alternate proposals, some of the text is also a little bit redundant. That will be corrected in future versions.

1.2. IDNA2008 and Special or Exceptional Cases

IDNA2008 contains several type of explicit provisions for characters (code points) that require special treatment when the requirements of the DNS cannot easily be met by calculations based on stable Unicode properties. Those provisions are
[[CREF1: ... to be supplied]]

As anticipated when IDNA2008, and RFC 5892 in particular, were written, exceptions and explicit updates are likely to be needed only if there is disagreement between the Unicode Consortium’s view about what is best for the Standard and its very diverse user community and the IETF’s view of what is best for IDNs, the DNS, and IDNA. It was hoped that a situation would never arise in which the the two perspectives would disagree, but the possibility was anticipated and considerable mechanism added to RFC 5890 and 5982 as a result. It is probably important to note that a disagreement in this context does not imply that anyone is "wrong", only that the two different groups have different needs and therefore criteria about what is acceptable. In particular, it appears that the Unicode Consortium has made assumptions about the availability (by explicit designation or context) of information about applicable languages or other context for a give string that are not possible for IDNA. For that reason, the IETF has, in the past, allowed some characters for IDNA that active Unicode Technical Committee members suggested be disallowed to avoid a change in derived tables [RFC6452]. This document describes a set of cases for which the IETF must consider disallowing sets of characters that the various properties would otherwise treat as PVALID.

This document provides the "flagging for the IESG" specified by Section 5.1 of RFC 5892. As specified there, the change itself requires IETF review because it alters the rules of Section 2 of that document.

> [[RFC Editor: please remove the following comment and note if they get to you.]]
>
> [[IESG: It might not be a bad idea to incorporate some version of the following into the Last Call announcement.]]
>
> NOTE IN DRAFT to IETF Reviewers: The issues in this document, and particularly the choices among options for either adding exception cases to RFC 5892 or ignoring the issue, warning people, and hoping the results do not include or enable serious problems, are fairly esoteric. Understanding them requires that one have at least some understanding of how scripts in which precomposed characters are preferred over combining sequences as a Unicode design and extension principle work. Those scripts include Arabic but, unlike the assumption when the issues were first discovered, are by no means limited to it. Readers should also understand the reasons the Unicode Standard gives various Arabic Script characters a fairly extended discussion [Unicode70-Arabic] but should treat that only as an example and note that most other cases are much less well documented. It also requires understanding of a number of Unicode principles, including the Normalization Stability rules [UAX15-Versioning] as applied to new precomposed characters and guidelines for adding new characters. There is considerable discussion of the issues in Section 3 and references are provided for those who want to pursue them, but potential reviewers should assume that the background needed to understand the reasons for this change is no less deep in the

subject matter than would be expected of someone reviewing a proposed change in, e.g., the fundamentals of BGP, TCP congestion control, or some cryptographic algorithm. Put more bluntly, one's ability to read or speak languages other than English, or even one or more languages that use the Arabic script or other scripts similarly affected, does not make one an expert in these matters.

1.3. Terminology

This document assumes that the reader is reasonably familiar with the terminology of IDNA [RFC5890] and Unicode [Unicode7] and with the IETF conventions for representing Unicode code points [RFC5137]. Some terms used here may not be used in the same way in those two sets of documents. From one point of view, those differences may have been the results of, or led to, misunderstandings that may, in turn, be part of the root cause of the problems explored in this document. In particular, this document uses the term "precomposed character" to describe characters that could reasonably be composed by a combining sequence using code points with appropriate appearance in common type styles but for which a single code point that does not require combining sequences is available. That definition is strictly about mechanical composition and does not involve any considerations about how the character is used. It is closely related to this document's definition of "identical". When a precomposed character exists and either applying NFC to the combining sequence does not yield that character or applying NFD to that character's code point does not yield the combining sequence, it is referred to in this document as "non-decomposable".

The document also uses some terms that are familiar to those who have been involved with IDNs and IDNA for a long time, but uses them more precisely than may be common in other quarters. For example, the term "Punycode" is not used at all in the rest of this document because it is the name of a very specific encoding algorithm [RFC3492] that does not incorporate the rules and algorithms for domain name labels that are produced by that encoding. Instead, the generic terms "ACE" or "ACE string" for "ASCII-compatible encoding" is used to refer to strings that abstractly contain characters outside the ASCII repertoire [RFC0020] but are encoded so that only ASCII characters appear in the string that would be encountered by a user or protocol and the terms "A-label" and "U-label", as defined in RFC 5890, to refer to the ACE and more conventional (or "native") character forms in which those non-ASCII characters appear in conventional Unicode encodings (typically UTF-8).

2. Document Aspirations

This document, in its present form, is not a proposal for a solution. Instead, it is intended to be (or evolve into) a comprehensive description of the issues and problems and to outline some possible approaches to a solution. A perfect solution -- one that would resolve all of the issues identified in this document -- would involve a relatively small set of relatively simple rules and hence would be comprehensible and predictable for and by non-expert end users, would not require code point by code point or even block by block exception lists, and would not leave uses of any script or language feeling that their particular writing system have been treated less fairly than others.

Part of the reality we need to accept is that IDNA, in its present form, represents compromises that does not completely satisfy those criteria and whatever is done about these issues will probably make it (or the job of administering zones containing IDNs) more complex. Similarly, as the Unicode Standard suggests when it identifies ten Design Principles and the text then says "Not all of these principles can be satisfied simultaneously..." [Unicode70-Design], while there are guidelines and principles, a certain amount of subjective judgment is involved in making determinations about normalization, decomposition, and some property values. For Unicode itself, those issues are resolved by multiple statements (at least one cited below) that one needs to rely on per-code point information in the Unicode Character Database rather than on rules or principles. The design of IDNA and the effort to keep it largely independent of Unicode versions requires rules, categories, and principles that can be relied upon and applied algorithmically. There is obviously some tension between the two approaches.

3. Problem Description

3.1. IDNA assumptions about Unicode normalization

IDNA makes several assumptions about Unicode, Unicode "characters", and the effects of normalization. Those assumptions were based on careful reading of the Unicode Standard at the time [Unicode5], guided by advice and commitments by members of the Unicode Technical Committee. Those assumptions, and the associated requirements, are necessitated by three properties of DNS labels that typically do not apply to blocks of running text:

1. There is no language context for a label. While particular DNS zones may impose restrictions, including language or script restrictions, on what labels can be registered, neither the DNS nor IDNA impose either type of restriction or give the user of a

label any indication about the registration or other restrictions that may have been imposed.

2. Labels are often mnemonics rather than words in any language. They may be abbreviations or acronyms or contain embedded digits and have other characteristics that are not typical of words.

3. Labels are, in practice, usually short. Even when they are the maximum length allowed by the DNS and IDNA, they are typically too short to provide significant context. Statements that suggest that languages can almost always be determined from relatively short paragraphs or equivalent bodies of text do not apply to DNS labels because of their typical short length and because, as noted above, they are not required to be formed according to language-based rules.

At the same time, because the DNS is an exact-match system, there must be no ambiguity about whether two labels are equal. Although there have been extensive discussions about "confusingly similar" characters, labels, and strings, such tests between scripts are always somewhat subjective: they are affected by choices of type styles and by what the user expects to see. In spite of the fact that the glyphs that represent many characters in different scripts are identical in appearance (e.g., basic Latin "a" (U+0061) and the identical-appearing Cyrillic character (U+0430), the most important test is that, if two glyphs are the same within a given script, they must represent the same character no matter how they are formed.

Unicode normalization, as explained in [UAX15], is expected to resolve those "same script, same glyph, different formation methods" issues. Within the Latin script, the code point sequence for lower case "o" (U+006F) and combining diaeresis (U+0308) will, when normalized using the "NFC" method required by IDNA, produce the precomposed small letter o with diaeresis (U+00F6) and hence the two ways of forming the character will compare equal (and the combining sequence is effectively prohibited from U-labels).

NFC was preferred over other normalization methods for IDNA because it is more compact, more likely to be produced on keyboards on which the relevant characters actually appeared, and because it does not lose substantive information (e.g., some types of compatibility equivalence involves judgment calls as to whether two characters are actually the same -- they may be "the same" in some contexts but not others -- while canonical equivalence is about different ways to produce the glyph for the same abstract character).

IDNA also assumed that the extensive Unicode stability rules would be applied and work as specified when new code points were added. Those rules, as described in The Unicode Standard and the normative annexes identified below, provide that:

1. New code points representing precomposed characters that can be formed from combining sequences will not be added to Unicode unless neither the relevant base character nor required combining character(s) are part of the Standard within the relevant script [UAX15-Versioning].

2. If circumstances require that principle be violated, normalization stability requires that the newly-added character decompose (even under NFC) to the previously-available combining sequence [UAX15-Exclusion].

At least at the time IDNA2008 was being developed, there was no explicit provision in the Standard’s discussion of conditions for adding new code points, nor of normalization stability, for an exception based on different languages using the same script or ambiguities about the shape or positioning of combining characters.

3.2. The discovery and the Arabic script cases

While the set of problems with normalization discussed above were discovered with a newly-added code point for the Arabic Script and some characteristics of Unicode handling of that script seem to make the problem more complex going forward, these are not issues specific to Arabic. This section describes the Arabic-specific problems; subsequent ones (starting with Section 3.3) discuss the problem more generally and include illustrations from other scripts.

3.2.1. New code point U+08A1, decomposition, and language dependency

Unicode 7.0.0 introduces the new code point U+08A1, ARABIC LETTER BEH WITH HAMZA ABOVE. As can be deduced from the name, it is visually identical to the glyph that can be formed from a combining sequence consisting of the code point for ARABIC LETTER BEH (U+0628) and the code point for Combining Hamza Above (U+0654). The two rules summarized above (see the last part of Section 3.1) suggest that either the new code point should not be allocated at all or that it should have a decomposition to \u’0628’\u’0654’.

Had the issues outlined in this document been better understood at the time, it probably would have been wise for RFC 5892 to disallow either the precomposed character or the combining sequence of each pair in those cases in which Unicode normalization rules do not cause the right thing to happen, i.e., the combining sequence and precomposed character to be treated as equivalent. Failure to do so at the time places an extra burden on registries to be sure that conflicts (and the potential for confusion and attacks) do not exist. Oddly, had the exclusion been made part of the specification at that time, the preference for precomposed forms noted above would probably have dictated excluding the combining sequence, something not otherwise done in IDNA2008 because the NFC requirement serves the same purpose. Today, the only thing that can be excluded without the potential disruption of disallowing a previously-PVALID combining sequence is the to exclude the newly-added code point so whatever is done, or might have been contemplated with hindsight, will be somewhat inconsistent.

3.2.2. Other examples of the same behavior within the Arabic Script

One of the things that complicates the issue with the new U+08A1 code point is that there are several other Arabic-script code points that behave in the same way for similar language-specific reasons.

In particular, at least three other grapheme clusters that have been present for many version of Unicode can be seen as involving issues similar to those for the newly-added ARABIC LETTER BEH WITH HAMZA ABOVE. ARABIC LETTER HAH WITH HAMZA ABOVE (U+0681) and ARABIC LETTER REH WITH HAMZA ABOVE (U+076C) do not have decomposition forms and are preferred over combining sequences using HAMZA ABOVE (U+0654) [Unicode70-Hamza]. By contrast, ARABIC LETTER ALEF WITH HAMZA ABOVE (U+0623) decomposes into \u'0627'\u'0654', ARABIC LETTER WAW WITH HAMZA ABOVE (U+0624) decomposes into \u'0648'\u'0654', and ARABIC LETTER YEH WITH HAMZA ABOVE (U+0626) decomposes into \u'064A'\u'0654' so the precomposed character and combining sequences compare equal when both are normalized, as this specification prefers.

There are other variations in which a precomposed character involving HAMZA ABOVE has a decomposition to a combining sequence that can form it. For example, ARABIC LETTER U WITH HAMZA ABOVE (U+0677) has a compatibility decomposition. but not a canonical one, into the combining sequence \u'06C7'\u'0674'.

3.2.3. Hamza and Combining Sequences

As the Unicode Standard points out at some length [Unicode70-Arabic], Hamza is a problematic abstract character and the "Hamza Above" construction even more so [Unicode70-Hamza]. Those sections explain a distinction made by Unicode between the use of a Hamza mark to denote a glottal stop and one used as a diacritic mark to denote a separate letter. In the first case, the combining sequence is used. In the second, a precomposed character is assigned.

Unlike Unicode generally and because of concerns about identifier spoofing and attacks based on similarities, character distinctions in

IDNA are based much more strictly on the appearance of characters; language and pronunciation distinctions within a script are not considered. So, for IDNA, BEH WITH HAMZA ABOVE is not-quite-tautologically the same as BEH WITH HAMZA ABOVE, even if one of them is written as U+08A1 (new to Unicode 7.0.0) and the other as the sequence \u'0628'\u'0654' (feasible with Unicode 7.0.0 but also available in versions of Unicode going back at least to the version [Unicode32] used in the original version of IDNA [RFC3490]. Because the precomposed form and combining sequence are, for IDNA purposes, the same, IDNA expects that normalization (specifically the requirement that all U-labels be in NFC form) will cause them to compare equal.

If Unicode also considered them the same, then the principle would apply that new precomposed ("composition") forms are not added unless one of the code points that could be used to construct it did not exist in an earlier version (and even then is discouraged) [UAX15-Versioning]. When exceptions are made, they are expected to conform to the rules and classes in the "Composition Exclusion Table", with class 2 being relevant to this case [UAX15-Exclusion]. That rule essentially requires that the normalization for the old combining sequence to itself be retained (for stability) but that the newly-added character be treated as canonically decomposable and decompose back to the older sequence even under NFC. That was not done for this particular case, presumably because of the distinction about pronunciation modifiers versus separate letters noted above. Because, for IDNA and the DNS, there is a possibility that the composing sequence \u'0628'\u'0654' already appears in labels, the only choice other than allowing an otherwise-identical, and identically-appearing, label with U+08A1 substituted to identify a different DNS entry is to DISALLOW the new character.

3.3. Precomposed characters without decompositions more generally

3.3.1. Description of the general problem

As mentioned above, IDNA made a strong assumption that, if there were two ways to form the same abstract character in the same script, normalization would result in them comparing equal. Work on IDNA2008 recognized that early version of Unicode might also contain some inconsistencies; see Section 3.3.2.3.2 below.

Having precomposed code points exist that don't have decompositions, or having code points of that nature allocated in the future, is problematic for those IDNA assumptions about character comparison. It seems to call for either excluding some set of code points that IDNA's rules do not now identify, development and use of a normalization procedure that behaves as expected (those two options may be nearly equivalent for many purposes), or deciding to accept a risk that, apparently, will only increase over time.

It is not clear whether the reasons the IDNABIS WG did not understand and allow for these cases are important except insofar as they inform considerations about what to do in the future. It seemed (and still seems to some people) that the Unicode Standard is very clear on the matter (or at least was when IDNA2008 was being developed). In addition to the normalization stability rules cited in the last part of Section 3.1. the discussion in the Core Standard seems quite clear. For example, "Where characters are used in different ways in different languages, the relevant properties are normally defined outside the Unicode Standard" in Section 2.2, subsection titled "Semantics" [Unicode7] did not suggest to most readers that sometimes separate code points would be allocated within a script based on language considerations. Similarly, the same section of the Standard says, in a subsection titled "Unification", "The Unicode Standard avoids duplicate encoding of characters by unifying them within scripts across language" and does not list exceptions to that rule or limit it to a single script although it goes on to list "CJK" as an example. Another subsection, "Equivalent Sequences" indicates "Common precomposed forms ... are included for compatibility with current standards. For static precomposed forms, the standard provides a mapping to an equivalent dynamically composed sequence of characters". The latter appears to be precisely the "all precomposed characters decompose into the relevant combining sequences if the relevant base and combining characters exist in the Standard" rule that IDNA needs and assumed and, again, there is no mention of exceptions, language-dependent of otherwise. The summary of stability policies cited in the Standard [Unicode70-Stability] does not appear to shed any additional light on these issues.

The Standard now contains a subsection titled "Non-decomposition of Overlaid Diacritics" [Unicode70-Overlay] that identifies a list of diacritics that do not normally form characters that have decompositions. The rule given has its own exceptions and the text clearly states that there is actually no way to know whether a code point has a decomposition other than consulting the Unicode Character Database entry for that code point. The subsequent section notes that this can be a security problem. While the issues with IDNA go well beyond what is normally considered security, that comment now seems clear. While that subsection is helpful in explaining the problem, especially for European scripts, it does not appear in the Unicode versions that were current when IDNA2008 was being developed.

3.3.2. Latin Examples and Cases

While this set of problems was discovered because of a code point added to the Arabic script in precombined form to support a particular language, there are actually far more examples for, e.g., Latin script than there are for Arabic script. Many of them are associated with the "non-decomposition of combining diacriticals" issues mentioned above, but the next subsections describe other cases that are not directly bound to decomposition.

3.3.2.1. The font exclusion and compatability relationships

Unicode contains a large collection of characters that are identified as "Mathematical Symbols". A large subset of them are basic or decorated Latin characters, differing from the ordinary ones only by their usage and, in appearance, by font or type styling (despite the general principle that font distinctions are not used as the basis for assigning separate code points. Most of these have canonical mappings to the base form, which eliminates them from IDNA, but others do not and, because the same marks that are used as phonetic diacritical markings in conventional alphabetical use have special mathematical meanings, applications that permit the use of these characters have their own issues with normalization and equality.

3.3.2.2. The phonetic notation characters and extensions

Another example involves various Phonetic Alphabet and Extension characters. many of which, unlike the Mathematical ones, do not have normalizations that would make them compare equal to the basic characters with essentially identical representations. This would not be a problem for IDNA if they were identified with a specialized script or as symbols rather than letters, but neither is the case: they are generally identified as lower case Latin Script letters even when they are visually upper-case, another issue for IDNA.

3.3.2.3. The stroke (solidus) ambiguity

Some combining characters have two or more forms. for example, in the case of the character popularly known as "slash", "stroke", or "solidus" (sometime prefixed by "forward"), there are "short" and "long" combining forms, U+0337 (COMBINING SHORT SOLIDUS OVERLAY) and U+0338 (COMBINING LONG SOLIDUS OVERLAY). It is not clear how long a short one needs to be to make it "long" or how short a long one needs to be to make it "short". Perhaps for that reason, U+00F8 has no decomposition and neither U+006F U+0337 nor U+006F U+0338 combine to it with NFC.

Adding to the confusion, at least when one attempts to use Unicode character names to identify places to look for problems, U+00F8 is formally called LATIN SMALL LETTER O WITH STROKE but, in combining character terminology, the term "stroke" refers to a horizontal bar, not an angled one, as in U+0335 and U+0336 (also short and long versions). However, when one overlays one of those on an "o" (U+006F), one gets U+0275, LATIN SMALL LETTER BARRED O, not "...o with stroke". That character, by the way, does not decompose either. This does illustrate the principle that it is not feasible to rely on Unicode code point names to identify confusable character sequences, even ones that produce the same, more or less font-independent, grapheme clusters.

3.3.2.3.1. Combining dots and other shapes combine... unless...

The discussion of "Non-decomposition of Overlaid Diacritics" [Unicode70-Overlay] indirectly exhibits at least one reason why it has been difficult to characterize the problem. If one combines that subsection with others, one gets a set of rules that might be described as:

1. If the precomposed character and the code points that make up the combining sequence exist, then canonical composition and decomposition work as expected, except...

2. If the precomposed character was added to Unicode after the code points that make up the combining sequence, normalization stability for the combining sequences requires that NFC applied to the precomposed character decomposes rather than having the combining sequence compose to the new character, however...

3. If the combining sequence involves a diacritic or other mark that actually touches the base character when composed, the precomposed character does not have a decomposition, unless...

4. The combining diacritic involved is Cedilla (U+0327), Ogonek (U+0328), or Horn (U+031B), in which case the precomposed characters that contain them "regularly" (but presumably not always) decomposes, and...

5. There are further exceptions for Hamza which does not overlay the associated base character in the same way the Latin-derived combining diacritics and other marks do. Those decisions to decompose a precomposed character (or not) are based on language or phonetic considerations, not the combining mechanism or appearance, or perhaps,...

6. Some characters have compatibility decompositions rather than canonical ones [Unicode70-CompatDecomp]. Because compatibility relationships are treated differently by IDNA, PRECIS [RFC8264], and, potentially, other protocols involving identifiers for Internet use, the existence of compatibility relationship may or may not be helpful. Finally,...

7. There is no reason to believe the above list is complete. In particular, if whether a precomposed character decomposes or not is determined by language or phonetic distinctions or by a decision that all new characters for some scripts will be precomposed while new ones for others will be added (if needed) as combining sequences, one may need additional rules on a per-script and/or per-character basis.

The above list only covers the cases involving combining sequences. It does not cover cases such as those in Section 3.3.2.1 and Section 3.3.2.2 and there may be additional groups of cases not yet identified.

3.3.2.3.2. "Legacy" characters and new additions

The development of categories and rules for IDNA recognized that early version of Unicode might contain some inconsistencies if evaluated using more contemporary rules about code point assignments and stability. In particular, there might be some exceptions from different practices in early version of Unicode or anomalies caused by copying existing single- or dual-script standards into Unicode as block rather than individual character additions to the repertoire. The possibility of such "legacy" exceptions was one reason why the IDNA category rules include explicit provisions for exception lists (even though no such code points were identified prior to 2014).

3.3.3. Unexpected Combining Sequances

Most combining characters have the script property "Inherited" or "Common", i.e., are not members of any particular script and will not cause rules against mixed-script labels to be triggered. Normalization rules are generally structured around the base character, so unexpected combinations of base characters with combining ones may lead to cases where normalization might normally be expected to produce a precombined character but does not do so (in the most common situation because no such precombined character exists. For example, the Latin script characters "a" and "a with acute accent" are both coded (as U+0061 and U+00E1). If the latter is coded as the combining sequence U+0061 U+0301, NFC will turn that sequence into U+00E1 and everything will work as users expect. However, the Cyrillic "a" character (U+0430) is notoriously similar in appearance in most type styles to U+0061 and the U+0439 U+0301 and that sequence does not normalize to anything else. Because thre is no code point assigned for Cyrillic small letter a with acute accent and unlike many of the other examples in this document, that is Unicode working exactly as would be expected. Whether it is an issue or not depends on the questions that are being asked and what rules are being applied.

3.3.4. Examples and Cases from Other Scripts

Research into these issues has not yet turned up a comprehensive list of affected scripts and code points. As discussed elsewhere in this document, it is clear that Arabic and Latin Scripts are significantly affected, that some Han and Kangxu radicals and ideographs are affected, and that other examples do exist -- it is just not known how many of those examples there are and what patterns, if any, characterize them.

3.3.4.1. Scripts with precomposed preferences and ones with combining preferences

While the authors have been unable to find an explanation for the differentiation in the Unicode Standard, we have been told that there are differences among scripts as to whether the action preference is to add new combining sequences only (and resist adding precomposed characters) as suggested in Section 3.3.2.3.1 or to add precomposed characters, often ones that do not have decompositions. If those difference in preference do exist, it is probably important to have them documented so that they can be reflected in IDNA review procedures and elsewhere. It will also require IETF discussion of whether combining sequences should be deprecated when the corresponding precomposed characters are added or to disallow combining sequences entirely for those scripts (as has been implicitly suggested for Arabic language use [RFC5564]).

[[CREF2: The above isn’t quite right and probably needs additional discussion and text.]]

3.3.4.2. The Han and Kangxu Cases

[[CREF3: .. to be supplied ..]]

3.4. Confusion and the Casual User

To the extent to which predictability for relatively casual users is a desired and important feather of relevant application or application support protocols, it is probably worth observing that the complex of rules and cases suggested or implied above is almost certainly too involved for the typical such user to develop a good intuitive understanding of how things behave and what relationships exist. Conversely, the nature of writing systems for natural languages, especially those that have evolved and diverged over centuries, implies that no set of rules about allowable characters will guarantee complete safety (however that is defined).

4. Implementation options and issues: Unicode properties, exceptions, and the nature of stability

4.1. Unicode Stability compared to IETF (and ICANN) Stability

The various stability rules in Unicode [Unicode70-Stability] all appear to be based on the model that once a value is assigned, it can never be changed. That is probably appropriate for a character coding system with multiple uses and applications. It is probably the only option when normative relationships are expressed in tables of values rather than by rules. One consequence of such a model is that it is difficult or impossible to fix mistakes (for some stability rules, the Unicode Standard does provide for exceptions) and even harder to make adjustments that would normally be dictated by evolution.

"No changes" provides a very strong and predictable type of stability. There are many reasons to take that path. As in some of the cases that motivated this document, the difficulty is that simply adding new code points (in Unicode) or features (in a protocol or application) may be destabilizing. One then has complete stability for systems that never use or allow the new code points or features, but rough edges for newer systems that see the discrepancies and rough edges. IDNA2003 (inadvertently) took that approach by freezing on Unicode 3.2 -- if no code points added after Unicode 3.2 had ever been allowed, we would have had complete stability even as Unicode libraries changed. Unicode has been quite ingenious about working around those difficulties with such provisions as having code points for newly-added precomposed characters decompose rather than altering the normalization for the combining sequences. Other cases, such as newly-added precomposed characters that do not decompose for, e.g., language or phonetic reasons, are more problematic.

The IETF (and ICANN and standards development bodies such as ISO and ISO/IEC JTC1) have generally adopted a different type of stability model, one which considers experience in use and the ill effects of not making changes as well as the disruptive effects of doing so. In the IETF model, if an earlier decision is causing sufficient harm and there is consensus in the communities that are most affected that a change is desirable enough to make transition costs acceptable, then the change is made.

The difference and its implications are perhaps best illustrated by a disagreement when IDNA2008 was being approved. IDNA2003 had effectively prevented some characters, notably (measured by intensity of the protests) the Sharp S character (U+00DF) from being used in DNS labels by mapping them to other characters before conversion to ACE form. It has also prohibited some other code points, notably ZWJ (U+200D) and ZWNJ (U+200C), by discarding them. In both cases, there were strong voices from the relevant language communities, supported by the registry communities, that the characters were important enough that it was more desirable to undergo the short-term pain of a transition and some uncertainty than to continue to exclude those characters and the IDNA2008 rules and repertoire are consistent with that preference. The Unicode Consortium apparently believed that stability --elimination of any possibility of label invalidation or different interpretations of the same string-- was more important than those writing system requirements and community preferences. That view was expressed through what was effectively a fork in (or attempt to nullify) the IETF Standard [UTS46] a result that has probably been worse for the overall Internet than either of the possible decision choices.

4.2. New Unicode Properties

One suggestion about the way out of these problems would be to create one or more new Unicode properties, maintained along with the rest of Unicode, and then incorporated into new or modified rules or categories in IDNA. Given the analysis in this document, it appears that that property (or properties) would need to provide:

1. Identification of combining characters that, when used in combining sequences, do not produce decomposable characters. [[CREF4: Wording on the above is not quite right but, for the present, maybe the intent is clear.]]

2. Identification of precomposed characters that might reasonably be expected to decompose, but that do not.

3. Identification of character forms that are distinct only because of language or phonetic distinctions within a script.

4. Identification of scripts for which precomposed forms are strongly preferred and combining sequences should either be viewed as temporary mechanisms until precomposed characters are assigned or banned entirely.

5. Identification of code points that represent symbols for specific, non-language, purposes even if identified as letters or numerals by their General Property. This would include all

characters given separate code points because of specialized "mathematical" and "phonetic" characters (see Section 3.3.2.2 and Section 3.3.2.1), but there are probably additional cases.

Some of these properties (or characteristics or values of a single property) would be suitable for disallowing characters, code points, or contextual sequences that otherwise might be allowed by IDNA. Others would be more suitable for making equality comparisons come out as needed by IDNA, particularly to eliminate distinctions based on language context.

While it would appear that appropriate rules and categories could be developed for IDNA (and, presumably, for PRECIS, etc.) if the problem areas are those identified in this document, it is not yet known whether the list is complete (and, hence, whether additional properties or information would be needed).

Even with such properties, IDNA would still almost certainly need exception lists. In addition, it is likely that stability rules for those properties would need to reflect IETF norms with arrangements for bringing the IETF and other communities into the discussion when tradeoffs are reviewed.

4.3. The need for exception lists

[[CREF5: Note in draft: this section is a partial placeholder and may need more elaboration.]]
Issues with exception lists and the requirements for them are discussed in Section 2 above and in RFC 5894 [RFC5894].

5. Proposed/ Alternative Changes to RFC 5892 for the issues first exposed by new code point U+08A1

NOTE IN DRAFT: See the comments in the Introduction, Section 1 and the first paragraph of each Subsection below for the status of the Subsections that follow. Each one, in combination with the material in Section 3 above, also provides information about the reasons why that particular strategy might or might not be appropriate.

When the term "Category" followed by an upper-case letter appears below, it is s reference to a rule in RFC 5892.

5.1. Disallow This New Code Point

This option is almost certainly too Arabic-specific and does not solve, or even address, the underlying problem. It also does not inherently generalize to non-decomposing precomposed code points that might be added in the future (whether to Arabic or other scripts) even though one could add more code points to Category F in the same way.

If chosen by the community, this subsection would update the portion of the IDNA2008 specification that identifies rules for what characters are permitted [RFC5892] to disallow that code point.

With the publication of this document, Section 2.6 ("Exceptions (F)") of RFC 5892 [RFC5892] is updated by adding 08A1 to the rule in Category F so that the rule itself reads:

```
F: cp is in {00B7, 00DF, 0375, 03C2, 05F3, 05F4, 0640, 0660,
             0661, 0662, 0663, 0664, 0665, 0666, 0667, 0668,
             0669, 06F0, 06F1, 06F2, 06F3, 06F4, 06F5, 06F6,
             06F7, 06F8, 06F9, 06FD, 06FE, 07FA, 08A1, 0F0B,
             3007, 302E, 302F, 3031, 3032, 3033, 3034, 3035,
             303B, 30FB}
```

and then add to the subtable designated
"DISALLOWED -- Would otherwise have been PVALID"
after the line that begins "07FA", the additional line:

```
08A1; DISALLOWED # ARABIC LETTER BEH WITH HAMZA ABOVE
```

This has the effect of making the cited code point DISALLOWED independent of application of the rest of the IDNA rule set to the current version of Unicode. Those wishing to create domain name labels containing Beh with Hamza Above may continue to use the sequence

U+0628, ARABIC LETTER BEH
followed by

U+0654, ARABIC HAMZA ABOVE

which was valid for IDNA purposes in Unicode 5.0 and earlier and which continues to be valid.

In principle, much the same thing could be accomplished by using the IDNA "BackwardCompatible" category (IDNA Category G, RFC 5892 Section 5.3). However, that category is described as applying only when "property values in versions of Unicode after 5.2 have changed in such a way that the derived property value would no longer be PVALID or DISALLOWED". Because U+08A1 is a newly-added code point in Unicode 7.0.0 and no property values of code points in prior versions have changed, category G does not apply. If that section of RFC 5892 were to be replaced in the future, perhaps consideration should be given to adding Normalization Stability and other issues to that description but, at present, it is not relevant.

5.2. Disallow This New Code Point and All Future Precomposed Additions that Do Not Decompose

At least in principle, the approach suggested above (Section 5.1) could be expanded to disallow all future allocations of non-decomposing precomposed characters. This would probably require either a new Unicode property to identify such characters and/or more emphasis on the manual, individual code point, checking of the new Unicode version review proces (i.e,. not just application of the existing rules and algorithm). It might require either a new rule in IDNA or a modification to the structure of Category F to make additions less tedious. It would do nothing for different ways to form identical characters within the same script that were not associated with decomposition and so would have to be used in conjunction with other appropaches. Finally, for scripts (such as Arabic) where there is a very strong preference to avoid combining sequences, this approach would exclude exactly the wrong set of characters.

5.3. Disallow the combining sequences for these characters

As in the approach discussed in Section 5.1, this approach is too Arabic-specific to address the more general problem. However, it illustrates a single-script approach and a possible mechanism for excluding combining sequences whose handling is connected to language information (information that, as discussed above, is not relevant to the DNS).

If chosen by the community, this subsection would update the portion of the IDNA2008 specification that identifies contextual rules [RFC5892] to prohibit (combining) Hamza Above (U+0654) in conjunction with Arabic BEH (U+0628), HAH (U+062D), and REH (U+0631). Note that the choice of this option is consistent with the general preference for precomposed characters discussed above but would ban some labels that are valid today and that might, in principle, be in use.

The required prohibition could be imposed by creating a new contextual rule in RFC 5892 to constrain combining sequences containing Hamza Above.

As the Unicode Standard points out at some length [Unicode70-Arabic], Hamza is a problematic abstract character and the "Hamza Above" construction even more so. IDNA has historically associated characters whose use is reasonable in some contexts but not others with the special derived property "CONTEXTO" and then specified specific, context-dependent, rules about where they may be used. Because Hamza Above is problematic (and spawns edge cases, as discussed in the Unicode Standard section cited above), it was suggested that a contextual rule might be appropriate. There are at least two reasons why a contextual rule would not be suitable for the present situation.

1. As discussed above, the present situation is a normalization stability and predictability problem, not a contextual one. Had the same issues arisen with a newly-added precomposed character that could previously be constructed from non-problematic base and combining characters, it would be even more clearly a normalization issue and, following the principles discussed there and particularly in UAX 15 [UAX15-Exclusion], might not have been assigned at all.

2. The contextual rule sets are designed around restricting the use of code points to a particular script or adjacent to particular characters within that script. Neither of these cases applies to the newly-added character even if one could imagine rules for the use of Hamza Above (U+0654) that would reflect the considerations of Chapter 8 of Unicode 6.2. Even had the latter been desired, it would be somewhat late now -- Hamza Above has been present as a combining character (U+0654) in many versions of Unicode. While that section of the Unicode Standard describes the issues, it does not provide actionable guidance about what to do about it for cases going forward or when visual identity is important.

5.4. Use Combinnig Classes to Develop Additional Contextual Rules

This option may not be of any practical use, but Unicode supports a property called "Combining_Class". That property has been used in IDNA only to construct a contextual rule for Zero-Width Non-Joiner [RFC5892, Appendix A.1] but speculation has arisen during discussions of work on Arabic combining characters and rendering [UTR53] as to whether Combining Classes could be used to build additional contextual rules that would restrict problematic cases. Unless such rules were applied only to new code points, they would also not be backward compatable.

The question of whether Combining Classes could be used to reduce the number of problematic labels is at least worth examination.

5.5. Disallow all Combining Characters for Specific Scripts

[[CREF6: This subsection needs to be turned into prose, but the follow bullet points are probably sufficient to identify the issues.]]

- Might work for Arabic and other "precomposed preference" scripts if those can be identified in an orderly and stable way (see Section 3.3.4.1; recommended by the Arabic language community for IDNs [RFC5564]).

- Unworkable for Latin because many characters that do not decompose are, at least in part, historical accidents resulting from combining prior national standards (this probably may exist for other scripts as well).

- No effect at all on special-use representations of identical characters within a script (see Section 3.3.2.1 and Section 3.3.2.2).

- Not backwards compatible.

5.6. Do Nothing Other Than Warn

A recommendation from UTC and others has been to simply warn registries, at all levels of the tree, to be careful with this set of characters. Doing that well would probably require making language distinctions within zones, which would violate the important IDNA principles that labels are not necessarily "words", do not carry language information, and may, at the protocol level, even deliberately mix languages and scripts. It is also problematic because the relevant set of characters is not easily defined in a precise way. This suggestion is problematic because the DNS and IDNA cannot make or enforce language distinctions, but it would avoid having the IETF either invalidate label strings that are potentially now in use or creating inconsistencies among the characters that combine with selected base characters but that also have precomposed forms that do not have decompositions. The potential would still exist for registries to respect the warning and deprecate such labels if they existed.

More generally, while there are already requirements in IDNA for registries to be knowledgeable and responsible about the labels they register (a separate document discusses that requirement [Klensin-rfc5891bis]), experience indicates that those requirements are often ignored. At least as important, warning registries about what should or should not be registered and even calling out specific code points as dangerous and in need of extra attention [Freytag-dangerous] does nothing to address the many cases in which lookup-time checking for IDNA conformance and deliberately misleading label constructions is important.

5.7. Normalization Form IETF (NFI))

The most radical possibility for the comparison issue would be to decide that none of the Unicode Normalization Forms specified in UAX 15 [UAX15] are adequate for use with the DNS because, contrary to their apparent descriptions, normalization tables are actually determined using language information. However, use of language information is unacceptable for IDNA for reasons described elsewhere in this document. The remedy would be to define an IETF-specific (or DNS-specific) normalization form (sometimes called "NFI" in discussions), building on NFC but adhering strictly to the rule that normalization causes two different forms of the same character (glyph image) within the same script to be treated as equal. In practice such a form could be implemented for IDNA purposes as an additional rule within RFC 5892 (and its successors) that constituted an exception list for the NFC tables. For this set of characters, the special IETF normalization form would be equivalent to the exclusion discussed in Section 5.3 above.

An Internet-identifier-specific normalization form, especially if specified somewhat separately from the IDNA core, would have a small marginal advantage over the other strategies in this section (or in combination with some of them), even though most of the end result and much of the implementation would be the same in practice. While the design of IDNA requires that strings be normalized as part of the process of determining label validity (and hence before either storage of values in the DNS or name resolution), there is an ongoing debate about whether normalization should be performed before storing a string or putting it on the wire or only when the string is actually compared or otherwise used.

If a normalization procedure with the right properties for the IETF was defined, that argument could be bypassed and the best decisions made for different circumstances. The separation would also allow better comparison of strings that lack language context in applications environments in which the additional processing and character classifications of IDNA and/or PRECIS were not applicable. Having such a normalization procedure defined outside IDNA would also minimize changes to IDNA itself, which is probably an advantage.

If the new normalizstion form were, in practice, simply an overlay on NFC with modifications dictated by exception and/or property lists, keeping its definition separate from IDNA would also avoid interweaving those exceptions and property lists with the rules and categories of IDNA itself, avoiding some unnecessary complexity.

6. Editorial clarification to RFC 5892

Verified RFC Editor Erratum 3312 [RFC5892Erratum] provides a clarification to Appendix A and Section A.1 of RFC 5892. This section of this document updates the RFC to apply that clarification.

1. In Appendix A, add a new paragraph after the paragraph that begins "The code point...". The new paragraph should read:

   "For the rule to be evaluated to True for the label, it MUST be evaluated separately for every occurrence of the Code point in the label; each of those evaluations must result in True."

2. In Appendix A, Section A.1, replace the "Rule Set" by

```
Rule Set:
  False;
  If Canonical_Combining_Class(Before(cp)) .eq.  Virama Then True;
  If cp .eq. \u200C And
         RegExpMatch((Joining_Type:{L,D})(Joining_Type:T)*cp
    (Joining_Type:T)*(Joining_Type:{R,D})) Then True;
```

7. Acknowledgements

The Unicode 7.0.0 changes were extensively discussed within the IAB’s Internationalization Program. The authors are grateful for the discussions and feedback there, especially from Andrew Sullivan and David Thaler. Additional information was requested and received from Mark Davis and Ken Whistler and while they probably do not agree with the necessity of excluding this code point or taking even more drastic action as their responsibility is to look at the Unicode Consortium requirements for stability, the decision would not have been possible without their input. Thanks to Bill McQuillan and Ted Hardie for reading versions of the document carefully enough to identify and report some confusing typographical errors. Several experts and reviewers who prefer to remain anonymous also provided helpful input and comments on preliminary versions of this document.

8. IANA Considerations

When the IANA registry and tables are updated to reflect Unicode 7.0.0, changes should be made according to the decisions the IETF makes about Section 5.

9. Security Considerations

From at least one point of view, this document is entirely a discussion of a security issue or set of such issues. While the "similar-looking characters" issue that has been a concern since the earliest days of IDNs [HomographAttack] and that has driven assorted "character confusion" projects [ICANN-VIP], if a user types in a string on one device and can get different results that do not compare equal when it is typed on a different device (with both behaving correctly and both keyboards appearing to be the same and for the same script) then all security mechanism that depend on the underlying identifiers, including the practical applications of DNS response integrity checks via DNSSEC [RFC4033] and DNS-embedded public key mechanisms [RFC6698], are at risk if different parties, at least one of them malicious, obtain or register some of the identical-appearing and identically-typed strings and get them into appropriate zones.

Mechanisms that depend on trusting registration systems (e.g., registries and registrars in the DNS IDN case, see Section 5.6 above) are likely to be of only limited utility because fully-qualified domains that may be perfectly reasonable at the first level or two of the DNS may have differences of this type deep in the tree, into levels where name management, and often accountability, are weak. Similar issues obviously apply when names are user-selected or unmanaged.

When the issue is not a deliberate attack but simple accidental confusion among similar strings, most of our strategies depend on the acceptability of false negatives on matching if there is low risk of false positives (see, for example, the discussion of false negatives in identifier comparison in Section 2.1 of RFC 6943 [RFC6943]). Aspects of that issue appear in, for example, RFC 3986 [RFC3986] and the PRECIS effort [RFC8264]. However, because the cases covered here are connected, not just to what the user sees but to what is typed and where, there is an increased risk of false positives (accidental as well as deliberate).

[[CREF7: Note in Draft: The paragraph that follows was written for a much earlier version of this document. It is obsolete, but is being retained as a placeholder for future developments.]]

This specification excludes a code point for which the Unicode-specified normalization behavior could result in two ways to form a visually-identical character within the same script not comparing equal. That behavior could create a dream case for someone intending to confuse the user by use of a domain name that looked identical to another one, was entirely in the same script, but was still considered different.

Internet Security in areas that involve internationalized identifiers that might contain the relevant characters is therefore significantly dependent on some effective resolution for the issues identified in this document, not just hand waving, devout wishes, or appointment of study committees about it.

10. References

10.1. Normative References

[RFC5137] Klensin, J., "ASCII Escaping of Unicode Characters", BCP 137, RFC 5137, DOI 10.17487/RFC5137, February 2008, <https://www.rfc-editor.org/info/rfc5137>.

[RFC5890] 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>.

[RFC5892] Faltstrom, P., Ed., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, DOI 10.17487/RFC5892, August 2010, <https://www.rfc-editor.org/info/rfc5892>.

[RFC5892Erratum] "RFC5892, "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", August 2010, Errata ID: 3312", Errata ID 3312, August 2012, <http://www.rfc-editor.org/errata_search.php?rfc=5892>.

[RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, DOI 10.17487/RFC5894, August 2010, <https://www.rfc-editor.org/info/rfc5894>.

[RFC6943] 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>.

[RFC8264] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 8264, DOI 10.17487/RFC8264, October 2017, <https://www.rfc-editor.org/info/rfc8264>.

[UAX15] Davis, M., Ed., "Unicode Standard Annex #15: Unicode Normalization Forms", June 2014, <http://www.unicode.org/reports/tr15/>.

[UAX15-Exclusion] "Unicode Standard Annex #15: ob. cit., Section 5", <http://www.unicode.org/reports/tr15/#Primary_Exclusion_List_Table>.

[UAX15-Versioning] "Unicode Standard Annex #15, ob. cit., Section 3", <http://www.unicode.org/reports/tr15/#Versioning>.

[Unicode5] The Unicode Consortium, "The Unicode Standard, Version 5.0", ISBN 0-321-48091-0, 2007.

Boston, MA, USA: Addison-Wesley. ISBN 0-321-48091-0. This printed reference has now been updated online to reflect additional code points. For code points, the reference at the time RFC 5890-5894 were published is to Unicode 5.2.

[Unicode62] The Unicode Consortium, "The Unicode Standard, Version 6.2.0", ISBN 978-1-936213-07-8, 2012, <http://www.unicode.org/versions/Unicode6.2.0/>.

Preferred citation: The Unicode Consortium. The Unicode Standard, Version 6.2.0, (Mountain View, CA: The Unicode Consortium, 2012. ISBN 978-1-936213-07-8)

[Unicode7] The Unicode Consortium, "The Unicode Standard, Version 7.0.0", ISBN 978-1-936213-09-2, 2014, <http://www.unicode.org/versions/Unicode7.0.0/>.

Preferred Citation: The Unicode Consortium. The Unicode Standard, Version 7.0.0, (Mountain View, CA: The Unicode Consortium, 2014. ISBN 978-1-936213-09-2)

[Unicode70-Arabic] "The Unicode Standard, Version 7.0.0, ob.cit., Chapter 9.2: Arabic", Chapter 9, 2014, <http://www.unicode.org/versions/Unicode7.0.0/ch09.pdf>.

Subsection titled "Encoding Principles", paragraph numbered 4, starting on page 362.

[Unicode70-CompatDecomp]
"The Unicode Standard, Version 7.0.0, ob.cit., Chapter 2.3: Compatibility Characters", Chapter 2, 2014, <http://www.unicode.org/versions/Unicode7.0.0/ch02.pdf>.

Subsection titled "Compatibility Decomposable Characters" starting on page 26.

[Unicode70-Design]
"The Unicode Standard, Version 7.0.0, ob.cit., Chapter 2.2: Unicode Design Principles", Chapter 2, 2014, <http://www.unicode.org/versions/Unicode7.0.0/ch02.pdf>.

[Unicode70-Hamza]
"The Unicode Standard, Version 7.0.0, ob.cit., Chapter 9.2: Arabic", Chapter 9, 2014, <http://www.unicode.org/versions/Unicode7.0.0/ch09.pdf>.

Subsection titled "Combining Hamza Above" starting on page 378.

[Unicode70-Overlay]
"The Unicode Standard, Version 7.0.0, ob.cit., Chapter 2.2: Unicode Design Principles", Chapter 2, 2014, <http://www.unicode.org/versions/Unicode7.0.0/ch02.pdf>.

Subsection titled "Non-decomposition of Overlaid Diacritics" starting on page 64.

[Unicode70-Stability]
"The Unicode Standard, Version 7.0.0, ob.cit., Chapter 2.2: Unicode Design Principles", Chapter 2, 2014, <http://www.unicode.org/versions/Unicode7.0.0/ch02.pdf>.

Subsection titled "Stability" starting on page 23 and containing a link to http://www.unicode.org/policies/stability_policy.html..

[UTS46] Davis, M. and M. Suignard, "Unicode Technical Standard #46: Unicode IDNA Compatibility Processing", Version 7.0.0, June 2014, <http://unicode.org/reports/tr46/>.

10.2. Informative References

[Dalby] Dalby, A., "Dictionary of Languages: The definitive reference to more than 400 languages", Columbia Univeristy Press , 2004.

pages 206-207

[Daniels] Daniels, P. and W. Bright, "The World's Writing Systems", Oxford University Press , 1986.

page 744

[Freytag-dangerous] Freytag, A., Klensin, J., and A. Sullivan, "Those Troublesome Characters: A Registry of Unicode Code Points Needing Special Consideration When Used in Network Identifiers", June 2017, <https://datatracker.ietf.org/doc/draft-freytag-troublesome-characters/>.

[HomographAttack] Gabrilovich, E. and A. Gontmakher, "The Homograph Attack", Communications of the ACM 45(2):128, February 2002, <http://www.cs.technion.ac.il/˜gabr/papers/homograph_full.pdf>.

[ICANN-VIP] ICANN, "The IDN Variant Issues Project: A Study of Issues Related to the Management of IDN Variant TLDs (Integrated Issues Report)", February 2012, <https://www.icann.org/en/system/files/files/idn-vip-integrated-issues-final-clean-20feb12-en.pdf>.

[Klensin-rfc5891bis] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Registry Restrictions and Recommendations", September 2017, <https://datatracker.ietf.org/doc/draft-klensin-idna-rfc5891bis/>.

[Omniglot-Fula] Ager, S., "Omniglot: Fula (Fulfulde, Pulaar, Pular'Fulaare)", <http://www.omniglot.com/writing/fula.htm>.

Captured 2015-01-07

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[RFC3490] 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>.

[RFC3492] 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>.

[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>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC5564] El-Sherbiny, A., Farah, M., Oueichek, I., and A. Al-Zoman, "Linguistic Guidelines for the Use of the Arabic Language in Internet Domains", RFC 5564, DOI 10.17487/RFC5564, February 2010, <https://www.rfc-editor.org/info/rfc5564>.

[RFC6452] Faltstrom, P., Ed. and P. Hoffman, Ed., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA) - Unicode 6.0", RFC 6452, DOI 10.17487/RFC6452, November 2011, <https://www.rfc-editor.org/info/rfc6452>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <https://www.rfc-editor.org/info/rfc6698>.

[Unicode32]
The Unicode Consortium, "The Unicode Standard, Version 3.2.0".

The Unicode Standard, Version 3.2.0 is defined by The Unicode Standard, Version 3.0 (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the Unicode Standard Annex #27: Unicode 3.1 (http://www.unicode.org/reports/tr27/) and by the Unicode Standard Annex #28: Unicode 3.2 (http://www.unicode.org/reports/tr28/).

[UTR53] Unicode Consortium, "Proposed Draft: Unicode Technical Report #53: Unicode Arabic Mark Ordering Algorithm", August 2017, <http://www.unicode.org/reports/tr53/>.

Note: this is a Proposed Draft, out for public review when this version of the current I-D is posted, and should not be considered either an approved/ final document or a stable reference.

Appendix A. Change Log

RFC Editor: Please remove this appendix before publication.

A.1. Changes from version -00 (2014-07-21)to -01

- Version 01 of this document is an extensive rewrite and reorganization, reflecting discussions with UTC members and adding three more options for discussion to the original proposal to simply disallow the new code point.

A.2. Changes from version -01 (2014-12-07) to -02

Corrected a typographical error in which Hamza Above was incorrectly listed with the wrong code point.

A.3. Changes from version -02 (2014-12-07) to -03

Corrected a typographical error in the Abstract in which RFC 5892 was incorrectly shown as 5982.

A.4. Changes from version -03 (2015-01-06) to -04

- Explicitly identified the applicability of U+08A1 with Fula and added references that discuss that language and how it is written.

- Updated several Unicode 6.2 references to point to Unicode 7.0 since the latter is now available in stable form (it was done when work on this I-D started).

- Extensively revised to discuss the non-Arabic cases, non-decomposing diacritics, other types of characters that don’t compare equal after normalization, and more general problem and approaches.

A.5. Changes from version -04 (2015-03-11) to -05

- Modified a few citation labels to make them more obvious.

- Restructured Section 1 and added additional terminology comments.

- Added discussion about non-decomposable character cases, including the "slash" example, and associated references for which -04 contained only placeholders.

- The examples and discussion of Latin script issues has been expanded considerably. It is unfortunate that many readers in the IETF community apparently cannot understand examples well enough to believe a problem is significant unless they is a discussion of Latin script examples, but, at least for this working draft, that is the way it is.

- Rewrote the discussion of several of the alternatives and added the discussion of combining classes.

- Rewrote and extended the discussion of the "warn only" alternative.

- Several other sections modified to improve technical or editorial clarity.

- Note that, while some references have been updated, others have not. In particular, Unicode references are still tied to versions 6 or 7. In some cases, those non-historical references are and will remain appropriate; others will best be replaced with information about current versions of documents.

Authors’ Addresses

John C Klensin
1770 Massachusetts Ave, Ste 322
Cambridge, MA 02140
USA

Phone: +1 617 245 1457
Email: john-ietf@jck.com

Patrik Faltstrom
Netnod
Franzengatan 5
Stockholm 112 51
Sweden

Phone: +46 70 6059051
Email: paf@netnod.se