IETF
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2018

A. Freytag
ASMUS, Inc.
J. Klensin

A. Sullivan
Oracle Corp.
June 29, 2018

Those Troublesome Characters: A Registry of Unicode Code Points Needing Special Consideration When Used in Network Identifiers
draft-freytag-troublesome-characters-02

Abstract

Unicode's design goal is to be the universal character set for all applications. The goal entails the inclusion of very large numbers of characters. It is also focused on written language in general; special provisions have always been needed for identifiers. The sheer size of the repertoire increases the possibility of accidental or intentional use of characters that can cause confusion among users, particularly where linguistic context is ambiguous, unavailable, or impossible to determine. A registry of code points that can be sometimes especially problematic may be useful to guide system administrators in setting parameters for allowable code points or combinations in an identifier system, and to aid applications in creating security aids for users.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 31, 2018.

Table of Contents

1. Unicode code points and identifiers

Unicode [Unicode] is a coded character set that aims to support every writing system. Writing systems evolve over time and are sometimes influenced by one another. As a result, Unicode encodes many characters that, to a reader, appear to be the same thing; but that are encoded differently from one another. This sort of difference is usually not important in written texts, because competent readers and writers of a language are able to compensate for the selection of the "wrong" character when reading or writing. Finally, the goal of supporting every writing system also implies that Unicode is designed to properly represent written language; special provisions are needed for identifiers.

Identifiers that are used in a network or, especially, an Internet context present several special problems because of the above feature of Unicode:

[[CREF1: AF: This whole business of language context seems unconnected from the data we have in the registry: that data is about code points and sequences that look the same, and many examples are in the same language. For example the duplicated shapes for digit / letter pairs. In very few cases would knowing the language context make a difference. In some cases, if you knew the script (not for the label, but the code point) you might be able to distinguish two labels, but that is it. I think we should further rewrite this summary so it matches better with the what the proposed registry contains.]]

1. In many (perhaps most) uses of identifiers, they are neither constrained to words in a particular language, nor would it be possible to ascertain reliably the language context in which the identifier is being or will be used. In the case of an internationalized domain name, for instance, each label could in principle represent a new locus of control, because there could be a delegation there. A new locus of control means that the administrator of the resulting zone could speak, read, or intend a different language context than the one from the parent. Moreover, at least some domains (such as the root) have an Internet-wide context and therefore do not really have a language context as such. In any case, the language context is simply not available as part of a DNS lookup, so there is no way to make the DNS sensitive to this sort of issue. Even in the case of email local-parts, where a sender is likely to know at least one of the languages of the receiver, the language context that was in use at the time the identifier was created is often unknown.

2. Identifiers on the network are in general exact-match systems, because an ambiguous identifier is problematic. Sometimes, but not always, there are facilities for aliasing such that multiple identifiers can be put together as a single identity; the DNS, for example, does not have such an aliasing capability, because in the DNS all aliases are one-way pointers. Aliasing techniques are in any case just an extension of the exact-match approach, and do not work the way a competent human reader does when interpolating the "right" character upon seeing the "wrong" one.

3. Because there are many characters that may appear to be the same (or even, that are defined in such a way that they are all but guaranteed to be rendered by the same glyphs), it is fairly easy to create an identifier either by accident or on purpose that is likely to be confused with some other identifier even by competent readers and writers of a language. In some cases knowing the language context would be of no help to recognition, for example, in cases where a language uses the same shape for a letter as for one of the digits.

4. For some scripts their repertoire of shapes overlaps with one or more other scripts, so that there are cases where two strings look identical to each other, even though all the code points in the first string are of one script, and all the code points in the second string are of another script. In these cases, the strings cannot be distinguished by a reader, and the whole strings are confusable.

5. For some scripts, both users and rendering systems do not expect to encounter code points in arbitrary sequence. Most code points normally occur only in specific locations within a syllable. If random labels were permitted, some would not display as expected (including having some features misplaced or not displayed) while others would present recognition problems to users experienced with the script. Some devices may also not support arbitrary input.

Beyond these issues, human perception is easily tricked, so that entirely unrelated character sequences can become confusable -- for example "rn" being confused with "m". Humans read strings, not characters, and they will mostly see what they expect to see. Some additional discussion of the background can be found in Appendix A.

The remainder of this document discusses techniques that can be used to design the label generation rules for a particular zone so they ameliorate or avoid entirely some of the issues caused by the interaction between the Unicode Standard and identifiers. The

registry is intended to highlight code points that require such techniques.

2. Background and Conventions

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

A reader needs to be familiar with Unicode [Unicode], IDNA2008 [RFC5890] [RFC5891] [RFC5892] [RFC5893] [RFC5894], PRECIS (at least the framework, [RFC7564]), and conventions for discussion of internationalization in the IETF (see [RFC6365]).

3. Techniques already in place

In the IDNA mechanism for including Unicode code points [RFC5892], a code point is only included when it meets the needs of internationalizing domain names as explained in the IDNA framework [RFC5894]. For identifiers other than those specified by IDNA, the PRECIS framework [RFC7564] generalizes the same basic technique. In both cases, the overall approach is to assume that all characters are excluded, and then to include characters according to properties derived from the Unicode character properties. This general strategy cuts the enormous size of the Unicode database somewhat, avoiding including some characters that are necessarily unsuited for use as identifiers.

The mechanism of inclusion by derived property, while helpful, is insufficient to guarantee every included character is safe for use in identifiers. Some characters' properties lead them to be included even though they are not obviously good candidates. In other cases, individual characters are good for inclusion, but are problematic in combination. Finally, there are cases where characters (or sequences of characters) are not problematic by themselves, or if used in a mutually exclusive manner in the same identifier, but become problematic when their choice represents the only difference between otherwise identical identifiers. For some examples, see Appendix B.

Operators of systems that create identifiers (whether through a registry or through a peer-to-peer identifier negotiation system) need to make policies for characters they will permit. Operators of registries, for instance, can help by adopting good registration policies: "Users will benefit if registries only permit characters from scripts that are well-understood by the registry or its advisers."[RFC5894]

The difficulty for many operators, however, is that they do not have the writing system expertise to claim any character is "well-understood", and they do not really have the time to develop that expertise. Such operators should in fact not use or register such characters. Unfortunately, in many cases the operators are stewards of systems where the user population demands identifiers useful to them in their local languages. In other cases, operators may proceed without a proper understanding owing to financial or market share incentives. The risk for Internet identifiers in such cases is obviously that ill-understood and potentially exploitable gaps in registration policies will open.

To help mitigate such issues, this document proposes a registry of Unicode code points that are known to present special issues for network identifiers with the aim to guide protocol and operating decisions about whether to permit a given code point or sequence of code points. By necessity, any list or guidance can only reflect issues that are known and understood at the time of writing. By limiting itself largely to characters that are widely used to write languages in contemporary use, the registry will address the more critical needs, while simultanesously focusing on characters that are well understood and for which there may already be some implementation experience in IDNs.

By itself, such a registry will not completely protect against poor registration or use, but it may provide operational guidance necessary for people who are responsible for creating policies. It also obviates the need for everyone to repeat basic investigation into the behavior of Unicode characters. Instead, scarce expertise can be focused on ways to mitigate issues, perhaps caused by user requirements for a specific character.

Note that the registry defined herein does not address any of the issues created by whole-string confusables where each of the identifiers is of a different script. A common workaround, limiting a registry to identifiers of only a single script, would mitigate this issue. [[CREF2: AF: we should evaluate that; cross-script variants that are homoglyphs have now been collected across modern scripts as part of the root zone LGR and are easily captured in a registry.]]

For some of the code points (or code point sequences) listed as presenting issues for identifiers, it may be most expeditious to simply not include them, even though they are valid according to the protocol. Sometimes, one of a pair of identical code points (or code point sequences) may be deemed preferable over the other for practical reasons.

However, simply leaving out any code point listed in this registry would render a registry of doubtful value for many scripts. It is not always necessary or desirable to exclude characters. Sometimes, it is merely necessary to ensure that for two otherwise identical identifiers, only one of a set of mutually exclusive code points (or sequences of code points) is used, while preventing the later registration of the label containing the other one in order to avoid ambiguity. This way the operator does not need to impose a choice.

In cases where two or more variants of such an identifier mean the same thing to the native reader, an operator may decide to allow all of the variant labels to be registered simultaneously, but only to the same entity (and with proper safeguards that limit the multiplicity of such allocatable variant labels).

The implementation of this strategy would be via the variant mechanism described in [RFC7940] and [RFC8228] which allows mechanical processing of mutual exclusion and /or bundling of identifiers respectively.

This specification defines a registry of code points and sequences that have been identified as requiring special attention when they are to be used in identifiers. An administrator who does not have the time or inclination to develop the requisite policies might contemplate simply not to permit these code points at all.

However, for some scripts the remaining subset might not be usable in a meaningful way. Identifiers in these scripts cannot be safely implemented without understanding the issues involved. Further note that many code points listed here are problematic only in their relationship to other code points and that as long as these issues are adequately addressed, for example using the variant mechanism, they do not need to be excluded. [[CREF3: AF: the above needs more editing, it’s a bit repetitive.]]

4. A registry of code points requiring special attention

4.1. Description

The registry contains four fields. [[CREF4: AF If we are limited to the "texttable" format, we are limited to three columns, there’s no way we can fit more than that into the RFC plain text format and remain legible. If we want more columns, then we need to use some other data format, including PDF (which would allow us to show the images for the code points).]]

1. The first field, called "Code Point(s)", is a code point or sequence of code points. Sequences in this and other fields are

listed as space separated code point values. For completeness, full code point sequences are listed, even if some of their constituents are "Not recommended". A code point value is a series of 4-6 uppercase hexadecimal digits, as defined in [Unicode].

2. The second field, "Related CP", contains zero or more cross references to related code points or sequences. Cross references consist of single code points or sequences. Multiple cross references are separated by a comma.

3. The third field, called "References", contains one or more references to documents describing the code point and the reason why it presents an issue. References are cited by numeric values, each in square brackets; multiple references are separated by space.

4. The last field, "Comment", is a free form text field that briefly describes the issue; it also The comment field starts with a category, separated by a colon, to allow quick identification of similar cases

The following are the defined category values:

Not Recommended While the code point (or sequence) is not DISALLOWED, there is emerging consensus in the community that it is not recommended for identifiers, or it is considered as such in the Unicode Standard. This includes, but is not limited to code points that are formally deprecated in the Unicode standard, as well as code points or sequences listed in the standard as "Do not use" or not preferred or similar. Code points not in active use, obsolete code points, or those intended for specialist use may also be listed under this category. Details are given in the explanation and references.

Identical The code point (or sequence) is normally identical in appearance to another code point (or sequence); or may be identical in some contexts. If the related CP is listed as "PREFERRED", it is recommended that this code point (or sequence) be excluded; in the case of a sequence, it may be appropriate to exclude, the constituent combining marks (after first consulting the details given in the listing for the marks). Otherwise, it is recommended to make the two identical code points or sequences mutually exclusive by treating them as variants. Details are given in the explanation and references.

Restricted Context The code point is problematic in relation to some other code points in the same label. For example, it should be

used only after some code points or not adjacent to certain other code points. Further details are given in the explanation and references. This is a common case for certain combining marks or other code points in so-called "complex" scripts. These scripts generally require a coordinated set of context rules; in those cases the registry would not list any specific context rules, but to point to documentation of existing Label Generation Rulesets implementing a coherent set of rules as examples. Code points with IDNA2008 property of CONTEXTJ or CONTEXTO are not listed, as long as the given context rules mitigate any concerns.

Preferred The code point is preferred to some other code point given in the cross reference (with the other code point normally "IDENTICAL" or "NOT RECOMMENDED"). In some cases this represents a preference for a code point (or sequence) that is a basic constituent in some alphabet over a code point (or sequence) that is rare or has specialized use. In some cases the preference may be formally specified or otherwise represent established community consensus. Details are given in the explanation and references.

Other All cases that do not fit one of the other categories. Details are given in the explanation and references.

If a character appears in the registry, that does not automatically mean that it is a bad candidate for use in identifiers generally. Absent a well-defined and verifiable policy, however, such a code point or sequence might well be treated with suspicion by users and by tools.

For code points tagged as being "identical" to or "indistinguishable" from other code points, it may be that one is preferred over the other, but it may also be that implementing a scheme for mutual exclusion of any resulting identical labels is the best solution, such as assigning them "blocked" variants according to [RFC7940] and [RFC8228].

Where characters are confusable with a combining sequence, only the combining sequence is listed; suggested mitigation may consist of disallowing either the specific combining sequence or disallowing the combining marks involved. It is usually inappropriate to exclude any of the basic letters involved, as they are generally members of the standard alphabet for one or more languages.

The registry and this document are to be understood as guidance for the purpose of developing operational policies that are used for protocols under normal administrative scope. For instance, zone operators that support IDNA are expected to create policies governing the code points that they will permit (see [RFC5894] and

[I-D.rfc5891bis]). The registry herein defined is intended to highlight particularly troublesome code points or code point sequences for the benefit of administrators creating such policies. It is also intended to highlight characters that may create identifier ambiguities and thereby create security vulnerabilities. However, by itself it is no substitute for such policies.

The registry is by necessity limited to code points for which adequate information is available; by and large this means code points used in connection with modern languages or writing systems, except that specialized extensions to modern scripts may be indicated, if their use would fall into any of the categories defined. Historic scripts, and any modern scripts not represented in the registry can be assumed to not be well-understood; operators are cautioned to locate other sources of information and to develop the necessary policies before deploying such scripts.

4.2. Maintenance

The registry is updated by Expert Review using an open process. From time to time, additional code points may be added to the Unicode standard, or further information may be discovered related to code points, to existing code points or those already listed here. The Unicode Standard may recommend against using a code point for all or some purposes. Or a script community may have gained more experience in deploying IDNs for that script and may create or update recommendations as to best policy.

4.3. Scope

Code points that are DISALLOWED in IDNA 2008 are not eligible to be listed. Code points that are CONTEXTJ or CONTEXTO are not included here unless there are documented concerns that are not mitigated by the existing IDNA context rules. The focus is on scripts that are significant for identifiers; code points from scripts that are historic or otherwise of limited use have generally not been considered - however exceptions may exist where authoritative information is readily available. Code points and code point sequences included are those that need special policies (including, but not limited to policies of exclusion).

New code points of sequences are listed whenever information becomes available that identifies a specific issue that requires attention in crafting a policy for the use of that code point or sequence in network identifiers. Likewise cross references, categories, explanations and references cited may be updated.

The contents of the registry generally does not represent original research but a collection of issues documented elsewhere, with appropriate references cited. An exception might be cases that are in clear analogy to existing entries, but not explicitly covered by existing references, for example, because the code point in question was recently added to Unicode.

If a particular language or script community reaches an apparent consensus that some code point is problematic, or that of two identical code points or sequences one should be preferred over the other, such recommendations, if known, should be documented in this registry.

In addition, if the Unicode Standard designates a code point as formally "deprecated" or less formally as "do not use", or identifies code points that are "intentionally identical", this is also something that should be reflected in the registry. Another source of potential information might be existing registry policies or recommended policies, particularly where it is apparent that they represent a careful analysis of the issue or a wider consensus, or both.

Proposed additions to the registry are to be shared on a mailing list to allow for broader comment and vetting.

If there is a disagreement about the existence of an issue or its severity, it is preferable to document both the issue and the different evaluations of it. In all cases, the information and documentation presented must allow a user to fully evaluate the status of any entry in the registry.

There is no requirement for the registry to form a stable body of data to which any future document would have to be backward compatible in any way. If new information emerges, additional code points may be considered problematic, or they may need to be reclassified. In case of significant changes, the explanation should note the nature of the change and cite a reference to document the basis for it.

5. Registry initial contents

5.1. Overview

IDNA 2008 uses an inclusion process based on Unicode properties to define which code points are PVALID, but also recognizes that some code points require a context rule (CONTEXTJ, CONTEXTO).

A number of code points which are PVALID in [RFC5892] may require additional attention in the design of label generations rules. In some cases, the issue is not necessarily with an individual code point, but with a code point sequence. In the following, "code point" and "code point sequence" are used synonymously unless explicitly called out. The fact that a code point require such attention does not affect its status under IDNA 2008.

The following describes a number of conditions that pose problems for network identifiers and common strategies for mitigating them.

5.2. Interchangeable Code Points

At times two code points or code point sequences are considered by all users (or a significant fraction) as equivalent to a degree that they accept one of them as substitute for another. This has obvious implications for the unambiguous recognition of identifiers. This document lists the code points and sequences affected (except for certain generic classes too numerous to list here). Note that one of the two may be preferred over the other, in which case the non-preferred one may be excluded or folded away. But in many cases either one is equally preferred. Mitigation techniques for such cases are discussed below.

Homoglyphs Homoglyphs are code points that have identical appearance, or are so close in appearance that they are indistinguishable if not presented side-by-side. Whenever two labels differ only by code points that are homoglyphs of each other and occur in the same position, users cannot distinguish the labels from each other or tell which label is intended, even though the underlying code points are different. Users will substitute one label for another.

Code points that are merely similar in appearance, including strongly similar code points, or code points that are difficult to distinguish (such as certain diacritical marks) are not considered here; handling such similarities often requires case by case judgment.

Instead, this document considers these types of code points that can be fully substituted for one another:

1. code points that, by design or derivation, are identical to each other;

2. code points that assume the same shape in some context, e.g. at the end of a label;

3. code points of a striking similarity based on derivation or common origin;

4. and code points that are otherwise indistinguishable from one another unless placed side by side.

Cross-script Homoglyphs A number of code points are homoglyphs of code points in another script (cross-script homoglyphs). Cross-script homoglyphs are a concern for any zone that supports labels from more than one script, even if each label is required to be in a single script. Note that some writing systems ordinarily use a combination of scripts (such as the use of Han, Hiragana and Katakana for Japanese). For many writing systems, an admixture of Latin letters is not uncommon, for example in brand or product names. If not handled carefully, this can prove problematic for identifiers.

Homophones As discussed in [202], the Amharic language treats many code points from the Ethiopic script as sound-alikes (homophones). In writing, these are freely substituted, users do not recognize some spelling as more correct. A conservative approach would treat these as mutually exclusive; the alternative, to make all variants available to the same applicant is appears not feasible due to the high number of such variants per label.

Semantic Variants The Chinese writing system, shared among several geographically distributed user communities, has many instances of code points that represent the same semantic. Even though they are visually distinct, they can be substituted for one another; typically these correspond to the simplified and traditional forms of Chinese characters. See [RFC4713] for details.

5.3. Excludable Code Points

Code points that are not substitutable but troublesome for other reasons are candidates for exclusion from a zone’s repertoire. For each such code point, the comment field briefly describes why it should be excluded or considered troublesome. There is no identified mitigation strategy that can be recommended for general usage: unless careful study indicates that a code point with this status is exceptionally acceptable for a particular zone, after all, it should normally be excluded from the repertoire. These reasons are varied.

Deprecated Code Points Deprecated code points are those that [Unicode] recommends not to use for any purpose. They should be excluded from identifiers; there is no mitigation. In addition, Unicode recommends against the use of some sequences and code

points for any purpose, but without formal deprecation. These should likewise be excluded from identifiers.

Non-preferred or other Troublesome Code Point This category includes all code points that are troublesome for other reasons; they include code points that represent non-preferred variations; or code points that not meant to be used in a combining sequence for letter; or code points that may be indistinguishable from a punctuation mark or other DISALLOWED code point. For each such code point, the comment field briefly describes why it should be excluded or considered troublesome.

Obsolete or not in Active Use Many code points across scripts that are otherwise in modern use represent additions for use in obsolete orthographies and writing systems, that is for writing languages that are extinct or not longer written in that script. Some have been researched and no evidence of active use could be found. These code points are not recommended for use in identifiers and should be excluded. Except for specialists, users are unlikely to recognize them, or find them of use in constructing mnemonic strings for identifiers. In addition, they often have not been sufficiently analyzed as to whether they represent other issues for identifiers. That makes their use risky. Obsolete, rare and code points otherwise not in active are generally not listed here. The reader can find a list of code points with high probability of being in active use in [MSR].

5.4. Combining Marks

Non Normalizable Sequences Certain combining marks are part of non-normalizable sequences. Normally, when a combining sequence is an alternate encoding to a composite code point, normalization can be used to select a preferred representation. For IDNA 2008, which uses NFC to normalize, this means the composite code point. However, some combining marks are not considered identical to the same mark when graphically part of a composite character. Sequences with these marks may look more or less like some composite code point, but they are considered different, and therefore not normalized. For identifiers, the best recommendation is to exclude those combining marks.

Combining marks that are also part of precomposed letters Many combining marks are part of canonical decompositions. For identifiers that are normalized to the composed forms using NFC (as required by IDNA 2008), these combining marks usually are not needed on their own, that is as separate element of a combining sequence after normalization. (The vast majority of letters using these marks have been encoded as precomposed characters). It is

strongly recommended to exclude these combining marks on their own, but, as needed for a specific language, to enumerate the needed sequences. (One notable example is Vietnamese which, after normalization to NFC uses a mixture of precomposed code points and combining marks). [TBD]The most common generic combining marks affected have been entered in the registry as excluded.

Non-spacing combining marks These marks are typically accents, diacritics and the like. They pose an additional problem: if they are allowed to occur twice in a row, some rendering systems will "overprint" them, in effect making them indistinguishable from single marks. This problem can be avoided by allowing only enumerated sequences, or alternatively by a context rule.

Ambiguous Rendering There are other ways in which certain code points and sequences representing particular combinations of code points may suffer from unreliable rendering, because rendering engines normally do not expect to encounter them. While Unicode allows the use of combining marks, in principle, in combination with any base character, in practice this can lead to unrecognizable labels, or labels that are not reliably distinct. This situation mostly affects the so-called complex scripts.

Combining marks in complex scripts In some scripts, there are no precomposed sequences. Usually, these scripts are "complex" scripts, that require context rules for many classes of code points. For these scripts, context rules (see [RFC7940]) should be used to limit non-spacing marks to acceptable contexts. For an example of such rules see [204], [206].

Soft Dotted and Dotless Letters Unicode code points with the Soft_Dotted property encode letter that lose their dot if followed by a diacritical mark above. (See [UCD]) If the following mark is a COMBINING DOT ABOVE, the combination is indistinguishable from the letter by itself. This can be mitigated by limiting or excluding the code point for DOT ABOVE. A soft dotted code point followed by any other diacritical mark above will look identical to the corresponding dotless letter with diacritical mark above. All combinations of dotless letters followed by diacritical marks should be excluded. (This can be done with a context rule, see [RFC7940]).

5.5. Mitigation

Thiere are several techniques that can be used to help to mitigate confusion. The focus in the following is on issues addressable by protocol or registry policy. However, user agents might implement

additional mitigation approaches, such as always using a font designed to distinguish among different characters.

5.5.1. Mitigation Strategies

Exclusion The primary mitigation technique is to reduce the problem space: 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. Users are unlikely to be familiar with many code points added to Unicode for the representation of historical forms of writing a script, or for highly specialized purposes. That unfamiliarity may present challenges to correct identification or keyboard entry, making the code point less usable. In addition, their use may present other problems not appreciated by anyone not familiar with them.

For these reasons, code points used only in a language with which the administrator is not familiar should probably be excluded. The same applies to code points used in specialized contexts, such as those only found in historic or sacred documents, or only used for phonetic transcription or poetry.

By reducing the repertoire to a well-understood essential subset it is often possible to eliminate some possible instances of confusion. For example, in the Arabic script, combining marks are generally used for optional or specialized aspects of the writing system. At the same time, many combining sequences are confusable with basic letters of the script. Because of this, excluding all Arabic combining mark would greatly reduce confusability without significantly affecting usability of the script for identifiers.

Preferred code points Sometimes, each of these code points will be used by a different user community; or one of the code points is not in wide use, for example because it is intended for special purposes like phonetic annotation or transliteration. In such cases, the one not needed for a given zone could be excluded.

In other cases, zones may be shared by a wider community, making it unattractive or impossible to institute a preference. A common method of mitigating issues from such homoglyphs is to make two labels that differ only by using a different homoglyph mutually exclusive. This can be done by making the homoglyphs code point variants, usually of type "blocked". See [RFC8228].

In some cases, while two code points may be homoglyphs, one of them can be identified as the preferred alternative to encode the intended character. In these cases, one of the code points has

been identified as "preferred", while the other has been identified as "troublesome"; or "excluded". In all other cases, no such preference exists in the general usage; a conservative mitigation might be to define the alternatives as blocked variants. However, the users of a given zone might have a specific preference, in which case one of the alternatives could be excluded instead.

For convenience in presentation, this document presents pairs or sets of homoglyphs as mutually exclusive variants of type "homoglyph". Other ways of handling these code points are possible. While one might implement such a variant relation in many cases as one label blocking another, in some cases allowing both to be registered to the same applicant may be appropriate. Finally, in some case eliminating one or both code points from the repertoire may be a feasible alternative to establishing a variant relation.

Script limitation For homoglyphs, a large number of cases (but not all of them) turn out to be in different scripts. As a result, it is usually a good idea to adopt the operational convention that identifiers for a protocol should always be in a single script.

This mitigation strategy has limits. First, even if any given identifier is only in a single script, it may co-exist with identifiers from other scripts. Sometimes the repertoire used in operation allows multiple scripts that create whole string confusables -- strings made up entirely of homoglyphs of another string in a different script (such as can be found between Cyrillic and Latin, for example). In such cases, mitigation must turn to other means of preventing the registration of mutually confusable string, for example by In that case, a robust mechanism for mutual exclusion of confusable identifiers must exist, ensuring that the registration of one of them (whichever comes first) blocks the later registration of the other.

Second, some writing systems use a combination of scripts and for commercial names in many scripts, admixture of Latin letters is common. Allowing limited script mixing may be an essential requirement in some cases.

Lastly, identifiers are not always under the operational control of a single authority (such as in the case of DNS, where the system is under distributed control so that different parts of the hierarchy can have different operational rules).

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.

Exact homoglyphs No policy or convention, other than ensuring mutual exclusion, will do anything to help mititgate confusion for strict homoglyphs of each other in the same script (see Appendix B for some example cases.)

Beyond the issue of mutual confusability, some combining sequences in particular can give rise to other difficulties in recognition - usually because client systems will not reliably and correctly display them. One particular case concerns sequences of more than one instance of the same non-spacing combining mark such as the repetition of an accent or diacritic. These are often rendered indistinguishably from single instances of the same mark. Operators should prohibit such repetition, particularly, as there are no known cases where they would be required in ordinary writing. Note that this prohibition would also apply to a non-spacing mark following a pre-composed code point containing the same diacritic. A more general mitigation technique would be to limit nonspacing marks to known combinations which can be enumerated. Where that is not possible for some scripts, some other context restrictions can usually be applied.

There are some writing systems where characters do not normally occur in arbitrary locations in the context of each syllable. Neither users nor rendering systems for such scripts are adept at handling arbitrary sequences of such characters. While some latitude beyond strict spelling rules may be accommodated, policies that enforce a minimal set of structural rules are required to ensure that users can identify the identifier and systems can render them predictably.

5.5.2. Limits of Mitigation

As noted in Section 1, it is not possible to solve all the problems with identifier systems, particularly when human factors are taken into account. In addition, each of the mitigation approaches has its own limits of the type of problems that can be addressed, whether it is by exclusion of specific code points; requiring or prohibiting contexts for certain code points; restriction to a single script per label; or mutual exclusion of labels differing only by code points identical or otherwise confusably equivalent to other code points. Additional policies may be needed to prevent registration of labels that are problematic or confusable for other reasons.

There are a number of issues in implementing and presenting identifiers to the user which are not specific to individually identifiable code points (or sequences). For example, fonts can vary widely in whether they make or do not make a distinction in appearance of characters; relying on the native reader to get the intended meaning from context. It is up to user agents to make sure to select fonts that render each code point as distinct as possible.

When new code points are assigned in Unicode, systems, keyboards, fonts and rendering engines may all be updated unevenly, with considerable delays. During a possibly lengthy transition period, this will lead to inconsistent user experience or inability to distinguish certain labels. Even if unsupported labels are presented as A-labels, users may not reliably identify them, because they appear as essentially random sequences of letters and digits.

5.6. Notes

In the explanation the character names have been abbreviated. The following list shows sample entries for the proposed registry. It is non-normative, and only included for illustrative purposes. Also see the examples below (Appendix B).

6. Table of Code Points

```
-----------------------------------------------------------------
  Code Point: 01C0
  Related CP:
  References: [120] [155]
  Comment:    Not Recommended: Indistinguishable from a
              punctuation character that is not PVALID
-----------------------------------------------------------------
  Code Point: 01C1
  Related CP:
  References: [120] [155]
  Comment:    Not Recommended: Indistinguishable from a
              punctuation character that is not PVALID
-----------------------------------------------------------------
  Code Point: 01C2
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from a
              punctuation character that is not PVALID
-----------------------------------------------------------------
  Code Point: 01C3
  Related CP:
  References: [120] [150]
  Comment:    Not Recommended: Indistinguishable from a
```

```
              punctuation character that is not PVALID
   ---------------------------------------------------------------
     Code Point: 01DD
     Related CP: 0259
     References: [150]
     Comment:    Identical: Identical in appearance to U+0259
   ---------------------------------------------------------------
     Code Point: 0259
     Related CP: 01DD
     References: [150]
     Comment:    Identical: Identical in appearance to U+01DD
   ---------------------------------------------------------------
     Code Point: 0131
     Related CP:
     References: [100]
     Comment:    Restricted Context: If followed by any combining
                 mark above, renders the same way as U+0069 in any
                 good font. Should be restricted to where it is not
                 followed by a combining mark above
   ---------------------------------------------------------------
     Code Point: 0237
     Related CP:
     References: [115]
     Comment:    Not Recommended: If followed by any combining mark
                 above, renders the same way as U+006A in any good
                 font. As its use is limited, it is best excluded.
   ---------------------------------------------------------------
     Code Point: 025F
     Related CP:
     References: [115]
     Comment:    Not Recommended: If followed by any combining mark
                 above, renders the same way as U+0249 in any good
                 font. As its use is limited, it is best excluded.
   ---------------------------------------------------------------
     Code Point: 02A3
     Related CP: 0064 007A
     References: [115]
     Comment:    Not Recommended: Looks like small LETTER D plus
                 LETTER Z, except for slight kerning; in limited
                 use.
   ---------------------------------------------------------------
     Code Point: 02A6
     Related CP: 0074 0073
     References: [115]
     Comment:    Not Recommended: Looks like small LETTER T plus
                 LETTER S, except for slight kerning; in limited
                 use.
   ---------------------------------------------------------------
```

```
  Code Point: 02A7
  Related CP: 0074 0283
  References: [115]
  Comment:    Not Recommended: Looks like small LETTER T plus
              LETTER ESH, except for slight kerning; in limited
              use.
---------------------------------------------------------------
  Code Point: 02AA
  Related CP: 006C 0073
  References: [115]
  Comment:    Not Recommended: Looks like small LETTER L plus
              LETTER S, except for slight kerning; in limited
              use.
---------------------------------------------------------------
  Code Point: 02AB
  Related CP: 006C 007A
  References: [115]
  Comment:    Not Recommended: Looks like small LETTER L plus
              LETTER Z, except for slight kerning; in limited
              use.
---------------------------------------------------------------
  Code Point: 02B9
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from a
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02BA
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from a
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02BB
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from a
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02BC
  Related CP:
  References: [6912]
  Comment:    Not Recommended: Indistinguishable from a
              punctuation character (U+2019), which is not
              PVALID
---------------------------------------------------------------
  Code Point: 02BD
  Related CP:
```

```
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02BE
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02BF
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02C0
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02C1
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02C6
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02C7
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02C8
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02C9
  Related CP:
```

```
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02CA
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 02CB
  Related CP:
  References: [120]
  Comment:    Not Recommended: Indistinguishable from
              punctuation character that is not PVALID
---------------------------------------------------------------
  Code Point: 0300
  Related CP:
  References: [100]
  Comment:    Not Recommended: Not recommended other than as
              part of enumerated sequences
---------------------------------------------------------------
  Code Point: 0301
  Related CP:
  References: [100]
  Comment:    Not Recommended: Not recommended other than as
              part of enumerated sequences
---------------------------------------------------------------
  Code Point: 0302
  Related CP:
  References: [100]
  Comment:    Not Recommended: Not recommended other than as
              part of enumerated sequences
---------------------------------------------------------------
  Code Point: 0303
  Related CP:
  References: [100]
  Comment:    Not Recommended: Not recommended other than as
              part of enumerated sequences
---------------------------------------------------------------
  Code Point: 0304
  Related CP:
  References: [100]
  Comment:    Not Recommended: Not recommended other than as
              part of enumerated sequences
---------------------------------------------------------------
  Code Point: 0306
  Related CP:
```

```
  References: [100]
  Comment:    Not Recommended: Not recommended other than as
              part of enumerated sequences
---------------------------------------------------------------
  Code Point: 0307
  Related CP:
  References: [115]
  Comment:    Restricted Context: By definition, LATIN SMALL
              LETTER I plus combining DOT ABOVE renders exactly
              the same as LATIN SMALL LETTER I by itself and
              does so in practice for any good font. The same is
              true for all Unicode characters with the
              soft_dotted property; they lose their dot if
              followed by a combining mark. DOT ABOVE should be
              excluded, or restricted to  contexts where it does
              not follow a soft_dotted letter.
---------------------------------------------------------------
  Code Point: 0308
  Related CP:
  References: [100]
  Comment:    Not Recommended: Not recommended other than as
              part of enumerated sequences
---------------------------------------------------------------
  Code Point: 0624
  Related CP: 0648
  References: [201]
  Comment:    Identical: Identical in appearance in some
              positional form and/or not reliably distinguished
              because of small size of distinguishing features
---------------------------------------------------------------
  Code Point: 0625
  Related CP: 0622, 0623, 0627, 0672
  References: [201]
  Comment:    Identical: Identical in appearance in some
              positional form and/or not reliably distinguished
              because of small size of distinguishing features
---------------------------------------------------------------
  Code Point: 0626
  Related CP: 0649, 064A, 067B, 06CC, 06CD, 06D0, 06D2
  References: [201]
  Comment:    Identical: Identical in appearance in some
              positional form and/or not reliably distinguished
              because of small size of distinguishing features
---------------------------------------------------------------
  Code Point: 0627
  Related CP: 0622, 0623, 0625, 0672
  References: [201]
  Comment:    Identical: Identical in appearance in some
```

positional form and/or not reliably distinguished because of small size of distinguishing features

---

Code Point: 064B
Related CP:
References: [5564]
Comment: Not Recommended: Not to be used in zone files for the Arabic language, per RFC 5564

Code Point: 064C
Related CP:
References: [5564]
Comment: Not Recommended: Not to be used in zone files for the Arabic language, per RFC 5564

Code Point: 065C
Related CP:
References: [300]
Comment: Not Recommended: Part of homoglyph sequence(s) not covered by normalization.

Code Point: 0660
Related CP: 06F0
References: [110]
Comment: Identical: Identical in appearance and meaning to EXTENDED ARABIC-INDIC DIGIT ZERO

Code Point: 0661
Related CP: 06F1
References: [110]
Comment: Identical: Identical in appearance and meaning to EXTENDED ARABIC-INDIC DIGIT ONE

Code Point: 077F
Related CP:
References: [115]
Comment: Not Recommended: Obsolote (archaic)

Code Point: 08AA
Related CP:
References: [201]
Comment: Not Recommended: No evidence of active use found; not recommended

---

Code Point: 0A72 0A3F
Related CP: 0A07
References: [401]
Comment: Not Recommended: Do not use for U+0A07

```
---------------------------------------------------------------
  Code Point: 0A72 0A40
  Related CP: 0A08
  References: [401]
  Comment:    Not Recommended: Do not use for U+0A08
---------------------------------------------------------------
  Code Point: 0E3A
  Related CP:
  References: [206]
  Comment:    Other issue: Renders unreliably, or not at all, if
              adjacent to any Thai vowel below. This may be
              prevented by a context rule
---------------------------------------------------------------
  Code Point: 0E41
  Related CP:
  References: [206]
  Comment:    Restricted Context: Digraph of U+0E40 SARA E
              U+0E40 SARA E. Normally handled by disallowing the
              sequence via a context rule
---------------------------------------------------------------
  Code Point: 0E45
  Related CP:
  References: [206]
  Comment:    Restricted Context: Only occurs after two special
              Thai vowels,U+0E24 RU and U+0E26 LU. Is also
              potentially confused with U+0E32 SARA I. Both
              issues can be addressed by defining a context
              rule. Alternatively the context may be spelled out
              by enumerating the two sequences and excluding
              U+0E45 if occurring by itself.
---------------------------------------------------------------
  Code Point: 0E4E
  Related CP:
  References: [206]
  Comment:    Not Recommended: Rarely used in modern Thai; it is
              more commonly replaced with U+0E3A (PHINTHU).
              Excluding it avoids issues with confusing it with
              another diacritic U+0E4C (THANTHAKHAT). Both are
              rendered atop a syllable and hard to distinguish
              at small sizes.
---------------------------------------------------------------
  Code Point: 12A5
  Related CP: 12D5
  References: [100] [202]
  Comment:    Interchangeable: U+12A5 and U+12D5 are used
              interchangeably in Amharic
---------------------------------------------------------------
  Code Point: 12A6
```

```
  Related CP: 12D6
  References: [100] [202]
  Comment:    Interchangeable: U+12A6 and U+12D6 are used
              interchangeably in Amharic
---------------------------------------------------------------
  Code Point: 17D2 178A
  Related CP: 17D2 178F
  References: [204]
  Comment:    Identical: When preceded by U+17D2, U+178A and
              U+178F are indistinguishable
---------------------------------------------------------------
  Code Point: 17D2 178F
  Related CP: 17D2 178A
  References: [204]
  Comment:    Identical: When preceded by U+17D2, U+178A and
              U+178F are indistinguishable
---------------------------------------------------------------
```

6.1. References for Registry

[99] The Unicode Consortium, "The Unicode Standard", (latest version) http:www.unicode.org/versions/latest (Multiple, or latest version)

[100] Integration Panel, "Maximal Starting Repertoire (MSR-2)", April 2015, https://www.icann.org/en/system/files/files/msr-2-overview-14apr15-en.pdf (Code points included in MSR-2 as potentially appropriate for the root zone)

[115] Integration Panel, "Maximal Starting Repertoire (MSR-2)", April 2015, https://www.icann.org/en/system/files/files/msr-2-overview-14apr15-en.pdf (Code points excluded from MSR-2 as inappropriate for the root zone)

[120] Integration Panel, "Maximal Starting Repertoire (MSR-2)", April 2015, https://www.icann.org/en/system/files/files/msr-2-overview-14apr15-en.pdf (Code points considered problematic by MSR-2)

[150] The Unicode Consortium, "Intentional.txt", Version 10.0.0, http://www.unicode.org/Public/security/10.0.0/intentional.txt (Code points considered identical by intention)

[155] "Proposal to Update Identical.txt", L2 17/301 (and revisions) http://www.unicode.org/L2/L2017/17301-update-intentional.pdf (Code points considered identical by intention)

[201] TF-AIDN, "Proposal for Arabic Script Root Zone LGR", 18 November 2015 https://www.icann.org/en/system/files/files/arabic-lgr-proposal-18nov15-en.pdf (In-script variants and code points excluded)

[202] Ethiopic Generation Panel, "Proposal for Ethiopic Script Root Zone LGR", May 17, 2017, https://www.icann.org/en/system/files/files/proposal-ethiopic-lgr-17may17-en.pdf ()

[204] Khmer Generation Panel, "Proposal for Khmer Script Root Zone Label Generation Rules (LGR)", August 15, 2016, https://www.icann.org/en/system/files/files/proposal-khmer-lgr-15aug16-en.pdf ()

[206] Thai Generation Panel, "Proposal for the Thai Script Root Zone LGR", May 25, 2017 https://www.icann.org/en/system/files/files/proposal-thai-lgr-25may17-en.pdf ()

[300] Internationalized Domain Names Variant Issues Project: Arabic Case Study Team Issues Report, ICANN, October 7, 2011 https://archive.icann.org/en/topics/new-gtlds/arabic-vip-issues-report-07oct11-en.pdf (In-script variants and code points excluded)

[401] Table 12-14 in Chapter 12 "South and Central Asia-I", ,"The Unicode Standard", Version 10.0, https://www.unicode.org/versions/Unicode10.0.0/ch12.pdf (Vowel sequences not to be used in Gurmukhi)

[5564] RFC 5564 (Code points to be excluded from repertoires for the Arabic language)

[6912] RFC 6912 (Code points considered problematic)

7. IANA Considerations

The IANA Services Operator is hereby requested to create the Registry of Unicode Code Points for Special Consideration in Network Identifiers, and to populate it with the values in section Section 5. The registry is to be updated by Expert Review.

This registry has no formal protocol status with respect to IDNA or PRECIS. It is a registry intended to be used by those creating registration or lookup policies, in order to inform the development of such policies.

8. Security Considerations

The registry established by this document is intended to help operators of identifier systems in deciding what to permit in identifiers. It may also be useful for user agents that attempt to provide warnings to users about suspicious or inadvisable identifiers. Operators that fail to make policies addressing the contents of the registry may permit the creation of identifiers that are misleading or that may be used in attacks on the network or users.

The registry is not a magic solution to all identifier ambiguity, and even refusing to permit registration of, or lookup of, every code point in the registry cannot ensure that misleading or confusing identifiers will never be created.

9. References

9.1. Normative References

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

[RFC4713] Lee, X., Mao, W., Chen, E., Hsu, N., and J. Klensin, "Registration and Administration Recommendations for Chinese Domain Names", RFC 4713, DOI 10.17487/RFC4713, October 2006, <https://www.rfc-editor.org/info/rfc4713>.

[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>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://www.rfc-editor.org/info/rfc5891>.

[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>.

[RFC5893] Alvestrand, H., Ed. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, DOI 10.17487/RFC5893, August 2010, <https://www.rfc-editor.org/info/rfc5893>.

[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>.

[RFC7564] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 7564, DOI 10.17487/RFC7564, May 2015, <https://www.rfc-editor.org/info/rfc7564>.

[RFC7940] Davies, K. and A. Freytag, "Representing Label Generation Rulesets Using XML", RFC 7940, DOI 10.17487/RFC7940, August 2016, <https://www.rfc-editor.org/info/rfc7940>.

[UAX44] The Unicode Consortium, "Unicode Standard Annex #44, Unicode Character Database", <http://www.unicode.org/reports/tr44/>.

This references the most currently published version of the description of the Unicode Character Database.

[UCD] The Unicode Consortium, "Unicode Character Database", <http://www.unicode.org/Public/UCD/latest/ucd/>.

This references the most currently published version of the data files for the Unicode Character Database

[Unicode] The Unicode Consortium, "The Unicode Standard, Latest Version", <http://www.unicode.org/versions/latest/>.

This references the most currently published version

9.2. Informative References

[I-D.klensin-idna-5892upd-unicode70]
Klensin, J. and P. Faltstrom, "IDNA Update for Unicode 7.0 and Later Versions", draft-klensin-idna-5892upd-unicode70-05 (work in progress), October 2017.

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

[MSR] Integration Panel, ""Maximal Starting Repertoire (MSR-3)"", March 2018, <https://www.icann.org/en/system/files/files/msr-3-overview-28mar18-en.pdf>.

[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>.

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

[RFC8228] Freytag, A., "Guidance on Designing Label Generation Rulesets (LGRs) Supporting Variant Labels", RFC 8228, DOI 10.17487/RFC8228, August 2017, <https://www.rfc-editor.org/info/rfc8228>.

[RZ-LGR] Integration Panel, ""Root Zone Label Generation Rules (LGR-2) - Overview and Summary"", July 2017, <https://www.icann.org/sites/default/files/lgr/lgr-2-overview-26jul17-en.pdf>.

Appendix A. Additional Background

A.1. The Theory of Inclusion

The mechanism that the IETF has come to prefer for internationalization of identifiers may be called "inclusion-based identifier internationalization", or "inclusion" for short. Under inclusion, the characters that are permissible in identifiers for a protocol are selected from the set of all Unicode characters. One starts with an empty set of characters, and then gradually adds characters to the set, usually based on Unicode properties (see below, and also Section 3).

Inclusion depends in part on assumptions the IETF made when the strategy was adopted and developed; some of those assumptions were about the relationships between different 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. Code points at issue are among those to be listed in the registry defined here. (See Section 5.)

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. For example, an e with an acute accent may be written by combining U+0065 LATIN SMALL LETTER E and U+0031 COMBINING ACUTE ACCENT, or it may be written U+00E9 LATIN SMALL LETTER E WITH ACUTE. 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 "canonical equivalence" between encodings of the same abstract characters is explicitly called out by Unicode. 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 (more precisely, 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, its direction when written, 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. They are defined in the Unicode Character Database [UCD] [UAX44].

Inclusion depends on the assumption that such strings as 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. If all the characters in the string are also included for the protocol’s candidate identifiers, then the string is eligible to be an identifier under the protocol.

A.2. The Difference Between Theory and Practice

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

Normalization Form C ("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 can 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.

A.2.1. Confusability

While Unicode deals in abstract characters and inclusion works on Unicode code points, users interact with strings as actually rendered: sequences of 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.)

Note that homoglyphs are not restricted to cross-script scenarios - there are a number of homoglyphs where both code points or sequences are part of the same script.

A further issue is introduced by the fact that Unicode caters not only to living and dead languages alike, but also to scholarly and scientific notation, as well as specialized modes of written text, such as for poetry, religious works, or texts to be sung or chanted. Where these notations use symbols, they are excluded under inclusion, but where they use varieties of letter forms or marks used with letters, they are included by default. Some of these letters or marks, have been incorporated over time into orthographies for living

languages, which is one reason they were not rigorously excluded from the start. However, in some cases, they may (alone or in combination with ordinary letters appear the same (or very similar to) existing letters. This makes some of these characters, and especially the marks in question "troublesome".

Finally, IDNA 2008 has a limited appreciation for the fact that characters in complex scripts, unlike ASCII letters, cannot simply occur in random sequences. Neither software (for display or data entering) nor readers are prepared to process some of these code points "out of order". For such scripts, without a policy that describes permissible contexts, labels could be registered that cannot be rendered or typed reliably and which most users would not know how to read or recognize. In some cases, combining sequences typed in the "wrong" order may display identically to to those typed in the "correct" ordering; again something that needs to be sorted out by defining permissible contexts, for example by using the context rule mechanism in [RFC7940].

Appendix B. 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 first notice the issue, even though the issue existed before this. 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 and similar Arabic combining marks are used are specialized enough that the combining marks are generally excluded. See [RFC5564] and [RZ-LGR]. Unicode has a policy of encoding as composite any letter needed in an Arabic orthography, even if it appears superficially that the same shape could be achieved by a combining sequence. (In actual typography there's often a small but noticeable difference in placement of the mark between a composite character and a combining sequence.)

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. Therefore, only the composite is used in IDNs.

U+09E1 vs u\’098C’u\’09E2’ This case is BENGALI LETTER VOCALIC LL. This is an example in the 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 signs in text, and the sequence of code points should not be used. There are similar cases in many Indic scripts. It is not a simple matter of disallowing the combining vowel mark in cases like this, because it is commonly used as vowel sign. The recommendation would be to add a context rule, restricting the vowel signs from appearing directly after an independent vowel like U+098C..

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 between the composite code point and the combining sequence, but only if one compares them side-by-side. Unlike a separable diacritic, there are no fast rules for placement of overlays. A bar may cross at different heights for different glyph shape or may cross different parts of the glyph. For this reason, there is no canonical equivalence defined between the sequence and the composite. Unicode has a principle of encoding barred letters of specific shape as single code point composites when needed for any writing system. The code point U+0335 COMBINING SHORT STROKE OVERLAY and similar overlay diacritics are therefore never needed as part of any orthography and are recommended to be excluded from identifiers.

U+00F8 vs \u’006F’\u’0337’ This is LATIN SMALL LETTER O WITH STROKE. The effect is 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 if viewed out of context. The use of the digraph is for specialized purposes; it can be excluded from identifiers.

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 inclusion and is therefore DISALLOWED in IDNA2008. 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. This an example that shows that the

Unicode name is not a reliable indicator of the intended appearance. Like other cases in Arabig, the recommendation is to exclude the combining mark (and therefore the sequence) in favor of the composite.

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 that demonstrate that the issue does not lie exclusively or primarily with combining sequences:

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 inclusion 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 a different member one of the seven identical shape pairs.

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

U+01DD vs U+0259 The two Latin script code points share the have the identical appearance of a lower-case upside down "e". They are encoded differently due to different uppercase forms. The fact that they uppercase differently is taken as evidence that they are not the same abstract character, despite the superficial evidence of their shared shape. The more common cases, where the uppercase forms are identical may be of less concern, given that IDNA 2008 is limited to lower case.

Cross script homoglyphs usually do not involve combining sequences, but can be mitigated by rules requiring strings to be in a single script. For zones that support multiple scripts, it may be necessary to have policies to prevent whole-script homographs: labels entirely in one script that look the same as another label in the other script. One method would be to define "blocked" variants (See [RFC7940] and [RFC8228]).

LATIN SMALL LETTER OPEN E is one of a handful of examples of characters borrowed from another script, in this case GREEK SMALL LETTER EPSILON.

LATIN SMALL LETTER E and CYRILLIC SMALL LETTER IE are historically related, both derive from uppercase forms of the GREEK CAPTIAL LETTER EPSILON. There are a number of such pairs -- enough to make many whole strings that look the same in both scripts (but usually spell nonsense in one of them). An example would be "pax".

Appendix C. Discussion Venue

Note to RFC Editor: this section should be removed prior to publication as an RFC.

This Internet-Draft may be discussed on the IAB Internationalization public list: i18n-discuss@iab.org.

Appendix D. Change History

Note to RFC Editor: this section should be removed prior to publication as an RFC.

00:

* Initial version

01:

* Add background and examples from the LUCID Problem Statement
* Add a paragraph about motivation to explain the difference between this registry and administrative policy more generally
* Expand and clarify a number of earlier points of discussion
* Attempt to make clear that this registry does not update any protocols

* Move some formerly-appendix material to the body

* Expand the initial registry.

02:

* Expanded the discussion of possible mitigation approaches and made its own section.

* Added more detail to the categories of troublesome characters

* Minor updates to "Existing techniques" section.

* Some extension to the description of the contents of the registry and discussion of how to handle additional information.

Authors' Addresses

Asmus Freytag
ASMUS, Inc.

Email: asmus@unicode.org

John C Klensin
1770 Massachusetts Ave, Ste 322
Cambridge, MA 02140
U.S.A.

Email: john-ietf@jck.com

Andrew Sullivan
Oracle Corp.
100 Milverton Drive
Missisauga, ON L5R 4H1
Canada

Email: andrew.s.sullivan@oracle.com