Network Working Group
Request for Comments: 3490
Category: Standards Track

P. Faltstrom
Cisco
P. Hoffman
IMC & VPNC
A. Costello
UC Berkeley

March 2003

Internationalizing Domain Names in Applications (IDNA)

Status of this Memo

Copyright Notice

Abstract

Until now, there has been no standard method for domain names to use characters outside the ASCII repertoire. This document defines internationalized domain names (IDNs) and a mechanism called Internationalizing Domain Names in Applications (IDNA) for handling them in a standard fashion. IDNs use characters drawn from a large repertoire (Unicode), but IDNA allows the non-ASCII characters to be represented using only the ASCII characters already allowed in so-called host names today. This backward-compatible representation is required in existing protocols like DNS, so that IDNs can be introduced with no changes to the existing infrastructure. IDNA is only meant for processing domain names, not free text.

Table of Contents

1. Introduction

IDNA works by allowing applications to use certain ASCII name labels (beginning with a special prefix) to represent non-ASCII name labels. Lower-layer protocols need not be aware of this; therefore IDNA does not depend on changes to any infrastructure. In particular, IDNA does not depend on any changes to DNS servers, resolvers, or protocol elements, because the ASCII name service provided by the existing DNS is entirely sufficient for IDNA.

This document does not require any applications to conform to IDNA, but applications can elect to use IDNA in order to support IDN while maintaining interoperability with existing infrastructure. If an application wants to use non-ASCII characters in domain names, IDNA is the only currently-defined option. Adding IDNA support to an existing application entails changes to the application only, and leaves room for flexibility in the user interface.

A great deal of the discussion of IDN solutions has focused on transition issues and how IDN will work in a world where not all of the components have been updated. Proposals that were not chosen by the IDN Working Group would depend on user applications, resolvers, and DNS servers being updated in order for a user to use an internationalized domain name. Rather than rely on widespread updating of all components, IDNA depends on updates to user applications only; no changes are needed to the DNS protocol or any DNS servers or the resolvers on user's computers.

1.1 Problem Statement

The IDNA specification solves the problem of extending the repertoire of characters that can be used in domain names to include the Unicode repertoire (with some restrictions).

IDNA does not extend the service offered by DNS to the applications. Instead, the applications (and, by implication, the users) continue to see an exact-match lookup service. Either there is a single exactly-matching name or there is no match. This model has served the existing applications well, but it requires, with or without internationalized domain names, that users know the exact spelling of the domain names that the users type into applications such as web browsers and mail user agents. The introduction of the larger repertoire of characters potentially makes the set of misspellings larger, especially given that in some cases the same appearance, for example on a business card, might visually match several Unicode code points or several sequences of code points.

IDNA allows the graceful introduction of IDNs not only by avoiding upgrades to existing infrastructure (such as DNS servers and mail transport agents), but also by allowing some rudimentary use of IDNs in applications by using the ASCII representation of the non-ASCII name labels. While such names are very user-unfriendly to read and type, and hence are not suitable for user input, they allow (for instance) replying to email and clicking on URLs even though the domain name displayed is incomprehensible to the user. In order to allow user-friendly input and output of the IDNs, the applications need to be modified to conform to this specification.

IDNA uses the Unicode character repertoire, which avoids the significant delays that would be inherent in waiting for a different and specific character set be defined for IDN purposes by some other standards developing organization.

1.2 Limitations of IDNA

The IDNA protocol does not solve all linguistic issues with users inputting names in different scripts. Many important language-based and script-based mappings are not covered in IDNA and need to be handled outside the protocol. For example, names that are entered in a mix of traditional and simplified Chinese characters will not be mapped to a single canonical name. Another example is Scandinavian names that are entered with U+00F6 (LATIN SMALL LETTER O WITH DIAERESIS) will not be mapped to U+00F8 (LATIN SMALL LETTER O WITH STROKE).

An example of an important issue that is not considered in detail in IDNA is how to provide a high probability that a user who is entering a domain name based on visual information (such as from a business card or billboard) or aural information (such as from a telephone or radio) would correctly enter the IDN. Similar issues exist for ASCII domain names, for example the possible visual confusion between the letter 'O' and the digit zero, but the introduction of the larger repertoire of characters creates more opportunities of similar looking and similar sounding names. Note that this is a complex issue relating to languages, input methods on computers, and so on. Furthermore, the kind of matching and searching necessary for a high probability of success would not fit the role of the DNS and its exact matching function.

1.3 Brief overview for application developers

Applications can use IDNA to support internationalized domain names anywhere that ASCII domain names are already supported, including DNS master files and resolver interfaces. (Applications can also define protocols and interfaces that support IDNs directly using non-ASCII representations. IDNA does not prescribe any particular representation for new protocols, but it still defines which names are valid and how they are compared.)

The IDNA protocol is contained completely within applications. It is not a client-server or peer-to-peer protocol: everything is done inside the application itself. When used with a DNS resolver library, IDNA is inserted as a "shim" between the application and the resolver library. When used for writing names into a DNS zone, IDNA is used just before the name is committed to the zone.

There are two operations described in section 4 of this document:

- The ToASCII operation is used before sending an IDN to something that expects ASCII names (such as a resolver) or writing an IDN into a place that expects ASCII names (such as a DNS master file).

- The ToUnicode operation is used when displaying names to users, for example names obtained from a DNS zone.

It is important to note that the ToASCII operation can fail. If it fails when processing a domain name, that domain name cannot be used as an internationalized domain name and the application has to have some method of dealing with this failure.

IDNA requires that implementations process input strings with Nameprep [NAMEPREP], which is a profile of Stringprep [STRINGPREP], and then with Punycode [PUNYCODE]. Implementations of IDNA MUST

fully implement Nameprep and Punycode; neither Nameprep nor Punycode are optional.

2. Terminology

The key words "MUST", "SHALL", "REQUIRED", "SHOULD", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

A code point is an integer value associated with a character in a coded character set.

Unicode [UNICODE] is a coded character set containing tens of thousands of characters. A single Unicode code point is denoted by "U+" followed by four to six hexadecimal digits, while a range of Unicode code points is denoted by two hexadecimal numbers separated by "..", with no prefixes.

ASCII means US-ASCII [USASCII], a coded character set containing 128 characters associated with code points in the range 0..7F. Unicode is an extension of ASCII: it includes all the ASCII characters and associates them with the same code points.

The term "LDH code points" is defined in this document to mean the code points associated with ASCII letters, digits, and the hyphen-minus; that is, U+002D, 30..39, 41..5A, and 61..7A. "LDH" is an abbreviation for "letters, digits, hyphen".

[STD13] talks about "domain names" and "host names", but many people use the terms interchangeably. Further, because [STD13] was not terribly clear, many people who are sure they know the exact definitions of each of these terms disagree on the definitions. In this document the term "domain name" is used in general. This document explicitly cites [STD3] whenever referring to the host name syntax restrictions defined therein.

A label is an individual part of a domain name. Labels are usually shown separated by dots; for example, the domain name "www.example.com" is composed of three labels: "www", "example", and "com". (The zero-length root label described in [STD13], which can be explicit as in "www.example.com." or implicit as in "www.example.com", is not considered a label in this specification.) IDNA extends the set of usable characters in labels that are text. For the rest of this document, the term "label" is shorthand for "text label", and "every label" means "every text label".

An "internationalized label" is a label to which the ToASCII operation (see section 4) can be applied without failing (with the UseSTD3ASCIIRules flag unset). This implies that every ASCII label that satisfies the [STD13] length restriction is an internationalized label. Therefore the term "internationalized label" is a generalization, embracing both old ASCII labels and new non-ASCII labels. Although most Unicode characters can appear in internationalized labels, ToASCII will fail for some input strings, and such strings are not valid internationalized labels.

An "internationalized domain name" (IDN) is a domain name in which every label is an internationalized label. This implies that every ASCII domain name is an IDN (which implies that it is possible for a name to be an IDN without it containing any non-ASCII characters). This document does not attempt to define an "internationalized host name". Just as has been the case with ASCII names, some DNS zone administrators may impose restrictions, beyond those imposed by DNS or IDNA, on the characters or strings that may be registered as labels in their zones. Such restrictions have no impact on the syntax or semantics of DNS protocol messages; a query for a name that matches no records will yield the same response regardless of the reason why it is not in the zone. Clients issuing queries or interpreting responses cannot be assumed to have any knowledge of zone-specific restrictions or conventions.

In IDNA, equivalence of labels is defined in terms of the ToASCII operation, which constructs an ASCII form for a given label, whether or not the label was already an ASCII label. Labels are defined to be equivalent if and only if their ASCII forms produced by ToASCII match using a case-insensitive ASCII comparison. ASCII labels already have a notion of equivalence: upper case and lower case are considered equivalent. The IDNA notion of equivalence is an extension of that older notion. Equivalent labels in IDNA are treated as alternate forms of the same label, just as "foo" and "Foo" are treated as alternate forms of the same label.

To allow internationalized labels to be handled by existing applications, IDNA uses an "ACE label" (ACE stands for ASCII Compatible Encoding). An ACE label is an internationalized label that can be rendered in ASCII and is equivalent to an internationalized label that cannot be rendered in ASCII. Given any internationalized label that cannot be rendered in ASCII, the ToASCII operation will convert it to an equivalent ACE label (whereas an ASCII label will be left unaltered by ToASCII). ACE labels are unsuitable for display to users. The ToUnicode operation will convert any label to an equivalent non-ACE label. In fact, an ACE label is formally defined to be any label that the ToUnicode operation would alter (whereas non-ACE labels are left unaltered by

ToUnicode). Every ACE label begins with the ACE prefix specified in section 5. The ToASCII and ToUnicode operations are specified in section 4.

The "ACE prefix" is defined in this document to be a string of ASCII characters that appears at the beginning of every ACE label. It is specified in section 5.

A "domain name slot" is defined in this document to be a protocol element or a function argument or a return value (and so on) explicitly designated for carrying a domain name. Examples of domain name slots include: the QNAME field of a DNS query; the name argument of the gethostbyname() library function; the part of an email address following the at-sign (@) in the From: field of an email message header; and the host portion of the URI in the src attribute of an HTML <IMG> tag. General text that just happens to contain a domain name is not a domain name slot; for example, a domain name appearing in the plain text body of an email message is not occupying a domain name slot.

An "IDN-aware domain name slot" is defined in this document to be a domain name slot explicitly designated for carrying an internationalized domain name as defined in this document. The designation may be static (for example, in the specification of the protocol or interface) or dynamic (for example, as a result of negotiation in an interactive session).

An "IDN-unaware domain name slot" is defined in this document to be any domain name slot that is not an IDN-aware domain name slot. Obviously, this includes any domain name slot whose specification predates IDNA.

3. Requirements and applicability

3.1 Requirements

IDNA conformance means adherence to the following four requirements:

1) Whenever dots are used as label separators, the following characters MUST be recognized as dots: U+002E (full stop), U+3002 (ideographic full stop), U+FF0E (fullwidth full stop), U+FF61 (halfwidth ideographic full stop).

2) Whenever a domain name is put into an IDN-unaware domain name slot (see section 2), it MUST contain only ASCII characters. Given an internationalized domain name (IDN), an equivalent domain name satisfying this requirement can be obtained by applying the

ToASCII operation (see section 4) to each label and, if dots are used as label separators, changing all the label separators to U+002E.

3) ACE labels obtained from domain name slots SHOULD be hidden from users when it is known that the environment can handle the non-ACE form, except when the ACE form is explicitly requested. When it is not known whether or not the environment can handle the non-ACE form, the application MAY use the non-ACE form (which might fail, such as by not being displayed properly), or it MAY use the ACE form (which will look unintelligle to the user). Given an internationalized domain name, an equivalent domain name containing no ACE labels can be obtained by applying the ToUnicode operation (see section 4) to each label. When requirements 2 and 3 both apply, requirement 2 takes precedence.

4) Whenever two labels are compared, they MUST be considered to match if and only if they are equivalent, that is, their ASCII forms (obtained by applying ToASCII) match using a case-insensitive ASCII comparison. Whenever two names are compared, they MUST be considered to match if and only if their corresponding labels match, regardless of whether the names use the same forms of label separators.

3.2 Applicability

IDNA is applicable to all domain names in all domain name slots except where it is explicitly excluded.

This implies that IDNA is applicable to many protocols that predate IDNA. Note that IDNs occupying domain name slots in those protocols MUST be in ASCII form (see section 3.1, requirement 2).

3.2.1. DNS resource records

IDNA does not apply to domain names in the NAME and RDATA fields of DNS resource records whose CLASS is not IN. This exclusion applies to every non-IN class, present and future, except where future standards override this exclusion by explicitly inviting the use of IDNA.

There are currently no other exclusions on the applicability of IDNA to DNS resource records; it depends entirely on the CLASS, and not on the TYPE. This will remain true, even as new types are defined, unless there is a compelling reason for a new type to complicate matters by imposing type-specific rules.

3.2.2. Non-domain-name data types stored in domain names

Although IDNA enables the representation of non-ASCII characters in domain names, that does not imply that IDNA enables the representation of non-ASCII characters in other data types that are stored in domain names. For example, an email address local part is sometimes stored in a domain label (hostmaster@example.com would be represented as hostmaster.example.com in the RDATA field of an SOA record). IDNA does not update the existing email standards, which allow only ASCII characters in local parts. Therefore, unless the email standards are revised to invite the use of IDNA for local parts, a domain label that holds the local part of an email address SHOULD NOT begin with the ACE prefix, and even if it does, it is to be interpreted literally as a local part that happens to begin with the ACE prefix.

4. Conversion operations

An application converts a domain name put into an IDN-unaware slot or displayed to a user. This section specifies the steps to perform in the conversion, and the ToASCII and ToUnicode operations.

The input to ToASCII or ToUnicode is a single label that is a sequence of Unicode code points (remember that all ASCII code points are also Unicode code points). If a domain name is represented using a character set other than Unicode or US-ASCII, it will first need to be transcoded to Unicode.

Starting from a whole domain name, the steps that an application takes to do the conversions are:

1) Decide whether the domain name is a "stored string" or a "query string" as described in [STRINGPREP]. If this conversion follows the "queries" rule from [STRINGPREP], set the flag called "AllowUnassigned".

2) Split the domain name into individual labels as described in section 3.1. The labels do not include the separator.

3) For each label, decide whether or not to enforce the restrictions on ASCII characters in host names [STD3]. (Applications already faced this choice before the introduction of IDNA, and can continue to make the decision the same way they always have; IDNA makes no new recommendations regarding this choice.) If the restrictions are to be enforced, set the flag called "UseSTD3ASCIIRules" for that label.

4) Process each label with either the ToASCII or the ToUnicode operation as appropriate. Typically, you use the ToASCII operation if you are about to put the name into an IDN-unaware slot, and you use the ToUnicode operation if you are displaying the name to a user; section 3.1 gives greater detail on the applicable requirements.

5) If ToASCII was applied in step 4 and dots are used as label separators, change all the label separators to U+002E (full stop).

The following two subsections define the ToASCII and ToUnicode operations that are used in step 4.

This description of the protocol uses specific procedure names, names of flags, and so on, in order to facilitate the specification of the protocol. These names, as well as the actual steps of the procedures, are not required of an implementation. In fact, any implementation which has the same external behavior as specified in this document conforms to this specification.

4.1 ToASCII

The ToASCII operation takes a sequence of Unicode code points that make up one label and transforms it into a sequence of code points in the ASCII range (0..7F). If ToASCII succeeds, the original sequence and the resulting sequence are equivalent labels.

It is important to note that the ToASCII operation can fail. ToASCII fails if any step of it fails. If any step of the ToASCII operation fails on any label in a domain name, that domain name MUST NOT be used as an internationalized domain name. The method for dealing with this failure is application-specific.

The inputs to ToASCII are a sequence of code points, the AllowUnassigned flag, and the UseSTD3ASCIIRules flag. The output of ToASCII is either a sequence of ASCII code points or a failure condition.

ToASCII never alters a sequence of code points that are all in the ASCII range to begin with (although it could fail). Applying the ToASCII operation multiple times has exactly the same effect as applying it just once.

ToASCII consists of the following steps:

1. If the sequence contains any code points outside the ASCII range (0..7F) then proceed to step 2, otherwise skip to step 3.

2. Perform the steps specified in [NAMEPREP] and fail if there is an error. The AllowUnassigned flag is used in [NAMEPREP].

3. If the UseSTD3ASCIIRules flag is set, then perform these checks:

   (a) Verify the absence of non-LDH ASCII code points; that is, the absence of 0..2C, 2E..2F, 3A..40, 5B..60, and 7B..7F.

   (b) Verify the absence of leading and trailing hyphen-minus; that is, the absence of U+002D at the beginning and end of the sequence.

4. If the sequence contains any code points outside the ASCII range (0..7F) then proceed to step 5, otherwise skip to step 8.

5. Verify that the sequence does NOT begin with the ACE prefix.

6. Encode the sequence using the encoding algorithm in [PUNYCODE] and fail if there is an error.

7. Prepend the ACE prefix.

8. Verify that the number of code points is in the range 1 to 63 inclusive.

4.2 ToUnicode

The ToUnicode operation takes a sequence of Unicode code points that make up one label and returns a sequence of Unicode code points. If the input sequence is a label in ACE form, then the result is an equivalent internationalized label that is not in ACE form, otherwise the original sequence is returned unaltered.

ToUnicode never fails. If any step fails, then the original input sequence is returned immediately in that step.

The ToUnicode output never contains more code points than its input. Note that the number of octets needed to represent a sequence of code points depends on the particular character encoding used.

The inputs to ToUnicode are a sequence of code points, the AllowUnassigned flag, and the UseSTD3ASCIIRules flag. The output of ToUnicode is always a sequence of Unicode code points.

1. If all code points in the sequence are in the ASCII range (0..7F) then skip to step 3.

2. Perform the steps specified in [NAMEPREP] and fail if there is an error. (If step 3 of ToASCII is also performed here, it will not affect the overall behavior of ToUnicode, but it is not necessary.) The AllowUnassigned flag is used in [NAMEPREP].

3. Verify that the sequence begins with the ACE prefix, and save a copy of the sequence.

4. Remove the ACE prefix.

5. Decode the sequence using the decoding algorithm in [PUNYCODE] and fail if there is an error. Save a copy of the result of this step.

6. Apply ToASCII.

7. Verify that the result of step 6 matches the saved copy from step 3, using a case-insensitive ASCII comparison.

8. Return the saved copy from step 5.

5. ACE prefix

The ACE prefix, used in the conversion operations (section 4), is two alphanumeric ASCII characters followed by two hyphen-minuses. It cannot be any of the prefixes already used in earlier documents, which includes the following: "bl--", "bq--", "dq--", "lq--", "mq--", "ra--", "wq--" and "zq--". The ToASCII and ToUnicode operations MUST recognize the ACE prefix in a case-insensitive manner.

The ACE prefix for IDNA is "xn--" or any capitalization thereof.

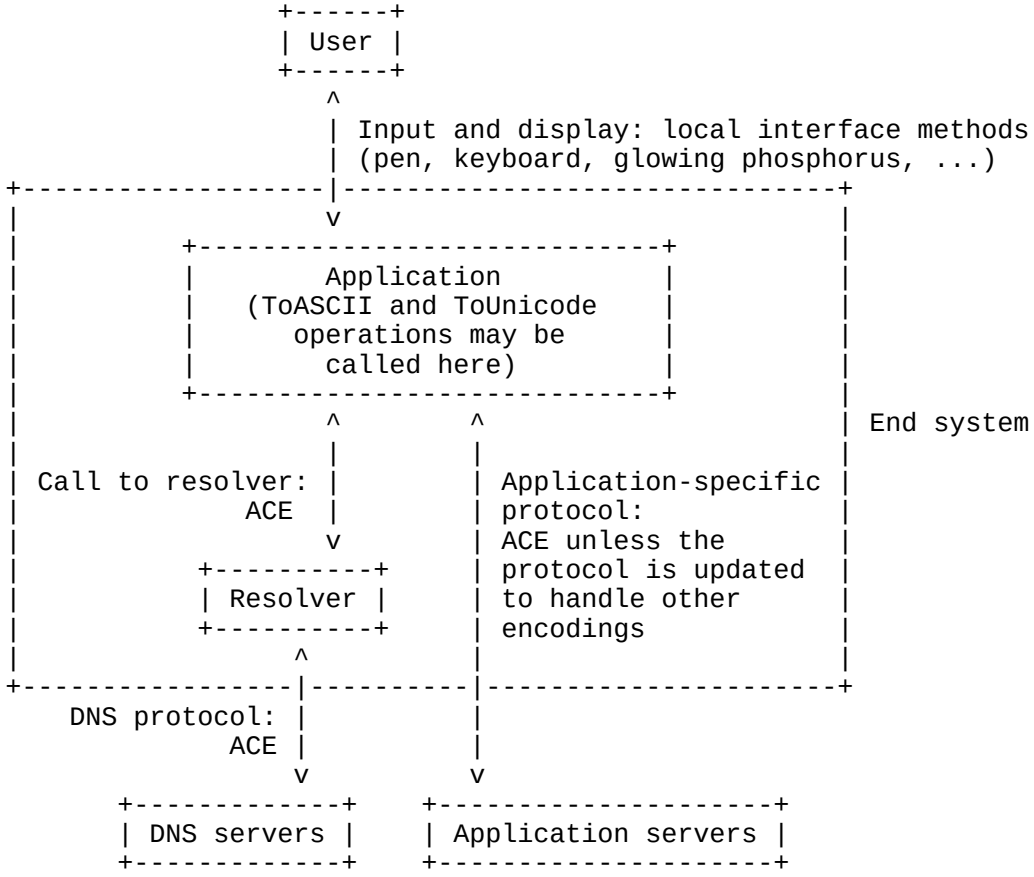
This means that an ACE label might be "xn--de-jg4avhby1noc0d", where "de-jg4avhby1noc0d" is the part of the ACE label that is generated by the encoding steps in [PUNYCODE].

While all ACE labels begin with the ACE prefix, not all labels beginning with the ACE prefix are necessarily ACE labels. Non-ACE labels that begin with the ACE prefix will confuse users and SHOULD NOT be allowed in DNS zones.

6. Implications for typical applications using DNS

In IDNA, applications perform the processing needed to input internationalized domain names from users, display internationalized domain names to users, and process the inputs and outputs from DNS and other protocols that carry domain names.

The components and interfaces between them can be represented pictorially as:



The box labeled "Application" is where the application splits a domain name into labels, sets the appropriate flags, and performs the ToASCII and ToUnicode operations. This is described in section 4.

6.1 Entry and display in applications

Applications can accept domain names using any character set or sets desired by the application developer, and can display domain names in any charset. That is, the IDNA protocol does not affect the interface between users and applications.

An IDNA-aware application can accept and display internationalized domain names in two formats: the internationalized character set(s) supported by the application, and as an ACE label. ACE labels that are displayed or input MUST always include the ACE prefix. Applications MAY allow input and display of ACE labels, but are not encouraged to do so except as an interface for special purposes, possibly for debugging, or to cope with display limitations as described in section 6.4.. ACE encoding is opaque and ugly, and should thus only be exposed to users who absolutely need it. Because name labels encoded as ACE name labels can be rendered either as the encoded ASCII characters or the proper decoded characters, the application MAY have an option for the user to select the preferred method of display; if it does, rendering the ACE SHOULD NOT be the default.

Domain names are often stored and transported in many places. For example, they are part of documents such as mail messages and web pages. They are transported in many parts of many protocols, such as both the control commands and the RFC 2822 body parts of SMTP, and the headers and the body content in HTTP. It is important to remember that domain names appear both in domain name slots and in the content that is passed over protocols.

In protocols and document formats that define how to handle specification or negotiation of charsets, labels can be encoded in any charset allowed by the protocol or document format. If a protocol or document format only allows one charset, the labels MUST be given in that charset.

In any place where a protocol or document format allows transmission of the characters in internationalized labels, internationalized labels SHOULD be transmitted using whatever character encoding and escape mechanism that the protocol or document format uses at that place.

All protocols that use domain name slots already have the capacity for handling domain names in the ASCII charset. Thus, ACE labels (internationalized labels that have been processed with the ToASCII operation) can inherently be handled by those protocols.

6.2 Applications and resolver libraries

Applications normally use functions in the operating system when they resolve DNS queries. Those functions in the operating system are often called "the resolver library", and the applications communicate with the resolver libraries through a programming interface (API).

Because these resolver libraries today expect only domain names in ASCII, applications MUST prepare labels that are passed to the resolver library using the ToASCII operation. Labels received from the resolver library contain only ASCII characters; internationalized labels that cannot be represented directly in ASCII use the ACE form. ACE labels always include the ACE prefix.

An operating system might have a set of libraries for performing the ToASCII operation. The input to such a library might be in one or more charsets that are used in applications (UTF-8 and UTF-16 are likely candidates for almost any operating system, and script-specific charsets are likely for localized operating systems).

IDNA-aware applications MUST be able to work with both non-internationalized labels (those that conform to [STD13] and [STD3]) and internationalized labels.

It is expected that new versions of the resolver libraries in the future will be able to accept domain names in other charsets than ASCII, and application developers might one day pass not only domain names in Unicode, but also in local script to a new API for the resolver libraries in the operating system. Thus the ToASCII and ToUnicode operations might be performed inside these new versions of the resolver libraries.

Domain names passed to resolvers or put into the question section of DNS requests follow the rules for "queries" from [STRINGPREP].

6.3 DNS servers

Domain names stored in zones follow the rules for "stored strings" from [STRINGPREP].

For internationalized labels that cannot be represented directly in ASCII, DNS servers MUST use the ACE form produced by the ToASCII operation. All IDNs served by DNS servers MUST contain only ASCII characters.

If a signaling system which makes negotiation possible between old and new DNS clients and servers is standardized in the future, the encoding of the query in the DNS protocol itself can be changed from

ACE to something else, such as UTF-8. The question whether or not this should be used is, however, a separate problem and is not discussed in this memo.

6.4 Avoiding exposing users to the raw ACE encoding

Any application that might show the user a domain name obtained from a domain name slot, such as from gethostbyaddr or part of a mail header, will need to be updated if it is to prevent users from seeing the ACE.

If an application decodes an ACE name using ToUnicode but cannot show all of the characters in the decoded name, such as if the name contains characters that the output system cannot display, the application SHOULD show the name in ACE format (which always includes the ACE prefix) instead of displaying the name with the replacement character (U+FFFD). This is to make it easier for the user to transfer the name correctly to other programs. Programs that by default show the ACE form when they cannot show all the characters in a name label SHOULD also have a mechanism to show the name that is produced by the ToUnicode operation with as many characters as possible and replacement characters in the positions where characters cannot be displayed.

The ToUnicode operation does not alter labels that are not valid ACE labels, even if they begin with the ACE prefix. After ToUnicode has been applied, if a label still begins with the ACE prefix, then it is not a valid ACE label, and is not equivalent to any of the intermediate Unicode strings constructed by ToUnicode.

6.5 DNSSEC authentication of IDN domain names

DNS Security [RFC2535] is a method for supplying cryptographic verification information along with DNS messages. Public Key Cryptography is used in conjunction with digital signatures to provide a means for a requester of domain information to authenticate the source of the data. This ensures that it can be traced back to a trusted source, either directly, or via a chain of trust linking the source of the information to the top of the DNS hierarchy.

IDNA specifies that all internationalized domain names served by DNS servers that cannot be represented directly in ASCII must use the ACE form produced by the ToASCII operation. This operation must be performed prior to a zone being signed by the private key for that zone. Because of this ordering, it is important to recognize that DNSSEC authenticates the ASCII domain name, not the Unicode form or

the mapping between the Unicode form and the ASCII form. In the presence of DNSSEC, this is the name that MUST be signed in the zone and MUST be validated against.

One consequence of this for sites deploying IDNA in the presence of DNSSEC is that any special purpose proxies or forwarders used to transform user input into IDNs must be earlier in the resolution flow than DNSSEC authenticating nameservers for DNSSEC to work.

7. Name server considerations

Existing DNS servers do not know the IDNA rules for handling non-ASCII forms of IDNs, and therefore need to be shielded from them. All existing channels through which names can enter a DNS server database (for example, master files [STD13] and DNS update messages [RFC2136]) are IDN-unaware because they predate IDNA, and therefore requirement 2 of section 3.1 of this document provides the needed shielding, by ensuring that internationalized domain names entering DNS server databases through such channels have already been converted to their equivalent ASCII forms.

It is imperative that there be only one ASCII encoding for a particular domain name. Because of the design of the ToASCII and ToUnicode operations, there are no ACE labels that decode to ASCII labels, and therefore name servers cannot contain multiple ASCII encodings of the same domain name.

[RFC2181] explicitly allows domain labels to contain octets beyond the ASCII range (0..7F), and this document does not change that. Note, however, that there is no defined interpretation of octets 80..FF as characters. If labels containing these octets are returned to applications, unpredictable behavior could result. The ASCII form defined by ToASCII is the only standard representation for internationalized labels in the current DNS protocol.

8. Root server considerations

IDNs are likely to be somewhat longer than current domain names, so the bandwidth needed by the root servers is likely to go up by a small amount. Also, queries and responses for IDNs will probably be somewhat longer than typical queries today, so more queries and responses may be forced to go to TCP instead of UDP.

9. References

9.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[STRINGPREP] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[NAMEPREP] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

[PUNYCODE] Costello, A., "Punycode: A Bootstring encoding of Unicode for use with Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

[STD3] Braden, R., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, and "Requirements for Internet Hosts -- Application and Support", STD 3, RFC 1123, October 1989.

[STD13] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034 and "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

9.2 Informative References

[RFC2535] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[UAX9] Unicode Standard Annex #9, The Bidirectional Algorithm, <http://www.unicode.org/unicode/reports/tr9/>.

[UNICODE] The Unicode Consortium. 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/).

[USASCII] Cerf, V., "ASCII format for Network Interchange", RFC 20, October 1969.

10. Security Considerations

Security on the Internet partly relies on the DNS. Thus, any change to the characteristics of the DNS can change the security of much of the Internet.

This memo describes an algorithm which encodes characters that are not valid according to STD3 and STD13 into octet values that are valid. No security issues such as string length increases or new allowed values are introduced by the encoding process or the use of these encoded values, apart from those introduced by the ACE encoding itself.

Domain names are used by users to identify and connect to Internet servers. The security of the Internet is compromised if a user entering a single internationalized name is connected to different servers based on different interpretations of the internationalized domain name.

When systems use local character sets other than ASCII and Unicode, this specification leaves the the problem of transcoding between the local character set and Unicode up to the application. If different applications (or different versions of one application) implement different transcoding rules, they could interpret the same name differently and contact different servers. This problem is not solved by security protocols like TLS that do not take local character sets into account.

Because this document normatively refers to [NAMEPREP], [PUNYCODE], and [STRINGPREP], it includes the security considerations from those documents as well.

If or when this specification is updated to use a more recent Unicode normalization table, the new normalization table will need to be compared with the old to spot backwards incompatible changes. If there are such changes, they will need to be handled somehow, or there will be security as well as operational implications. Methods to handle the conflicts could include keeping the old normalization, or taking care of the conflicting characters by operational means, or some other method.

Implementations MUST NOT use more recent normalization tables than the one referenced from this document, even though more recent tables may be provided by operating systems. If an application is unsure of which version of the normalization tables are in the operating

system, the application needs to include the normalization tables itself. Using normalization tables other than the one referenced from this specification could have security and operational implications.

To help prevent confusion between characters that are visually similar, it is suggested that implementations provide visual indications where a domain name contains multiple scripts. Such mechanisms can also be used to show when a name contains a mixture of simplified and traditional Chinese characters, or to distinguish zero and one from O and l. DNS zone adminstrators may impose restrictions (subject to the limitations in section 2) that try to minimize homographs.

Domain names (or portions of them) are sometimes compared against a set of privileged or anti-privileged domains. In such situations it is especially important that the comparisons be done properly, as specified in section 3.1 requirement 4. For labels already in ASCII form, the proper comparison reduces to the same case-insensitive ASCII comparison that has always been used for ASCII labels.

The introduction of IDNA means that any existing labels that start with the ACE prefix and would be altered by ToUnicode will automatically be ACE labels, and will be considered equivalent to non-ASCII labels, whether or not that was the intent of the zone adminstrator or registrant.

11. IANA Considerations

IANA has assigned the ACE prefix in consultation with the IESG.

12. Authors' Addresses

Patrik Faltstrom
Cisco Systems
Arstaangsvagen 31 J
S-117 43 Stockholm Sweden

EMail: paf@cisco.com

Paul Hoffman
Internet Mail Consortium and VPN Consortium
127 Segre Place
Santa Cruz, CA 95060 USA

EMail: phoffman@imc.org

Adam M. Costello
University of California, Berkeley

URL: http://www.nicemice.net/amc/

13. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.