The ISP Column
An occasional column on things Internet

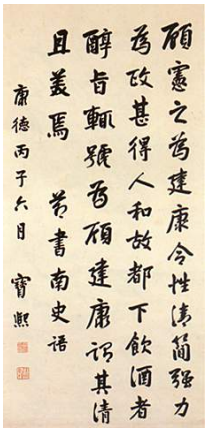
December 2006

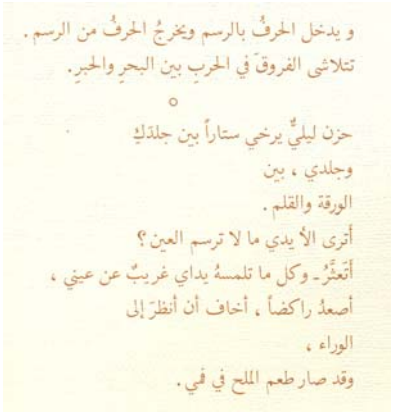
Geoff Huston

Internationalizing the Internet

Considering the global reach of the Internet, it sounds like a tautology to consider the issues of internationalizing the network. Surely it already reaches around the globe to every country, doesn't it? How much more "international" can you get? But maybe I'm just being too parochial here when I call it a tautology. I use a dialect of the English language, and all the characters I need are contained in the Western Latin character set. Because of this, I have avoided the experience of using a non-English language on the Internet. For me, the only language I use on the Internet is English, and all the characters I need are encompassed in the ASCII character set. If I were to attempt to use the Internet via a language that uses a non-Latin character set and a different script then maybe my experience would be different and most likely, an acutely frustrating one at that! If I were a native speaker of an entirely different language to English, then I'd probably give the Internet a low score for ease of use. While its easy to see picture of words in a variety of fonts and scripts, using them in an intuitive and natural way in the context of the Internet becomes an altogether more challenging problem.

Mostly what's needed is good *localization*, or adapting the local computing environment to suit local linguistic needs. This may include support for additional character sets, additional language scripts, and even altering the direction of symbol flow.





A few years back I had a technical book that I wrote (in English) translated into Japanese. I was somewhat surprised to see that the book used a word flow direction similar to English (left to right word flow, top to bottom word rows, left to right pagination), as I had thought that Japanese language printed material used a top-to-bottom word flow then right-to-left columns and right to left pagination.

"Traditionally, Japanese is written in a format called *tategaki*. In this format, the characters are written in columns going from top to bottom, with columns ordered from right to left. After reaching the bottom of each column, the reader continues at the top of the column to the left of the current one. This copies the

column order of Chinese. Modern Japanese also uses another writing format, called *yokogaki*. This writing format is identical to that of European languages such as English, with characters arranged in rows which are read from left to right, with successive rows going downwards."

[http://en.wikipedia.org/wiki/Japanese_script]

"Today, the left-to-right direction is dominant in all three languages [Chinese characters, Japanese kana, and Korean hangul] for horizontal writing: this is due partly to the influence of English, and partly to the increased use of computerized typesetting and word processing software, most of which does not directly support right-to-left layout of East Asian languages."

[http://en.wikipedia.org/wiki/Horizontal_and_vertical_writing_in_East_Asian_scripts]

So it would appear that even *yokogaki* is an outcome of lack of capability of our IT systems to undertake localization correctly!

One topic, however, does not appear to have a compellingly obvious *localization* solution in this multi-lingual environment, and that is the Domain Name System (DNS). The subtle difference here is that the DNS is the glue that binds all users' language symbols together, and performing localized adaptations to suit local language use needs is not enough. What we need is a means to allow all of these language symbols to be used within the same system, or *internationalization*.

The DNS is the most prevalent means of initiating a network transaction, whether it's a torrent session, the web, email, or any other form of network activity. But the DNS name string is not just an arbitrary string of characters. What you find in the DNS is most often a sequence of words or their abbreviations, and the words are generally English words, using characters drawn from a subset of the Latin character set. Perhaps unsurprisingly, some implementations of the DNS also assume that all DNS names can only be constructed from this ASCII character set and are incapable of supporting a larger character repertoire. So if you want to use a larger character set in order to represent various diacritics, such as acute and grave symbols, umlauts and similar, then the deployed DNS can be resistant to this use and provide incorrect responses to queries that attempt to use a larger character repertoire than plain old alpha-numerics. And if you want to use words drawn from languages that do not use the western script for their characters, such as Japanese or Thai, for example, then the DNS is highly resistant to this form of intended use.

Latin and Roman alphabets

The default Latin alphabet is the Roman alphabet , supplemented with G, J, U, W, Y, Z, and lower-case variants

Additional letters may be formed:
- as ligatures, as W was from VV, for example Æ (*ash*) from AE, *oethel* Œ from OE, *eszett* ß from ſz (long s + z), *engma* ŋ from NG, *ou* ȣ from OU, Ñ from NN, or ä from ae;
- by diacritics, such as Å, Č, Ų;
- as digraphs, such as IJ and LL;
- by modification, as J was from I, G from C, Ø from O, *eth* Ð from D, *yogh* ȝ from G, *schwa* Ə from E, or *ezh* ʒ from z;
- may even be borrowed from another alphabet entirely, as *thorn* Þ and *wynn* Ƿ were from Futhark [Runic].

[http://en.wikipedia.org/wiki/Roman_script]

One of the few global communications networks that has had to cope with aspects of this multi-lingual problem in the past has been the international postal network.

[From: "Tintin in Tibet", Herge, 1960, http://en.wikipedia.org/wiki/Tintin_in_Tibet]

Their form of resolution is not exactly encouraging news, in that if you want your letter to be delivered you may well need to use a bilingual address on the envelope, or, minimally, put in enough information on the envelope in the local language and local script to get the envelope into the right outgoing mail bag, and also use the language and script of the destination to provide the specific intended destination address. And if this is not possible, then try using the English language and the Latin character set and hope that your letter makes it through the system!

Although, to their credit, the postal system did manage to solve their version of the third party transit problem: "One of the most important results of the UPU [Universal Postal Union] treaty was that it ceased to be necessary, as it often had been previously, to affix the stamps of any country through which one's letter or package would pass in transit; the UPU provides that stamps of member nations are accepted for the whole international route."

[http://en.wikipedia.org/wiki/Universal_Postal_Union]

So lets restate the problem from the perspective of an end user of the Internet. I am a native English speaker, well the Australian dialect of English at any rate, living in a location where English is the predominant local language, and I use an unadorned version of the Latin character alphabet for writing. So for me the Internet presents a natural experience most of the time. But if I wanted to use a different language, and in particular a language that was not based on the Latin character set, would I have the same ease of use?

The computing industry has done a reasonable job of at least presenting non-Latin-based scripts within many applications and while at times it appears to represent a less than reasonable compromise, it is possible to enter non-Latin characters on keyboards. So it is possible to customise my local environment to use a language other than English in a relatively natural way.

But what about multi-lingual support in the wider world of the Internet?

Again the overall story is not all that bad. Non-Latin character scripts can be used in email, in all kinds of web documents, including this one, and in a wide variety of network applications. We can tag content with a language context to allow the content to be displayed in the correct language using the appropriate character sets and presentation glyphs. However, until recently, one area continued to stick steadfastly to its ASCII roots, and that was the Domain Name System. In this article I'd like to look at this topic of internationalization of the DNS, or Internationalized Domain Names, or IDNs.

So what do we mean when we talk of "internationalizing the DNS"?

What is being referred to here is an environment where English, and the Latin character set, is just one of many languages and scripts in use, and where a communication is initiated in one locale then the language and presentation is preserved irrespective of the locale where the communication is received.

Terminology

To start off on this topic there is a small amount of terminology that is helpful to get through:

Language	A language uses characters drawn from a collection of scripts
Script	A collection of characters that are related in their use by a language
Character	A unit of a script
Glyph	The presentation of a character within the style of a font
Font	A collection of glyphs encompassing a script's character set that share a consistent presentation style

Multiple languages can use a common script, and any locale or country may use many languages, reflecting the diversity of its population and the evolution of local dialects within communities.

It is also useful to bear in mind the distinction between *internationalization* and *localization*. *Internationalization* is concerned with providing a common substrate that can be used by many, preferably all, languages and all users, while *localization* is concerned with the use of a particular language within a particular locale and within a defined user population. Unsurprisingly the two concepts are often confused, particularly when true internationalization is often far harder to achieve than localization.

Internationalizing the DNS

The objective is the internationalization of the DNS, such that the DNS becomes capable of supporting the union of all character sets, while preserving the absence of ambiguity and uncertainty in terms of resolution of any individual DNS name. What is needed is to take a description of all possible characters in all languages, and allow their use in the DNS. So the starting point is the "universal character set", and that appears to be Unicode.

One of the basic building blocks for internationalization is the use of a character set which is the effective union of all character sets. Unicode (www.unicode.org) is intended to be such a universal encoding of characters (and symbols) in the contexts of all scripts and all languages. The current version of the Unicode standard, version 5.0, contains 98,884 distinct coded graphic characters.

A sequence of Unicode code points can be represented in multiple ways by using different character encoding schemes. The most commonly used schemes are UTF-8 and UTF-16.

UTF-8 is a variable-length encoding using 8 bit words, which means that different code points require different numbers of bytes. The larger a code point's index number, the more bytes required to represent it using UTF-8. For example, the first 127 Unicode code points, which correspond exactly to the values used by the ASCII character set (which maps only 127 characters), can be represented using only 8 bits in UTF-8, using the same 8 bit values as in ASCII. UTF-8 can require up to 32 bits to encode certain code points. A criticism of UTF-8 is that it 'penalizes' certain scripts by requiring more bytes to represent those scripts' code points. The IETF has made the UTF-8 its preferred default character encoding for internationalization of Internet application protocols.

UTF-16 is a variable length character encoding using 16 bit words. Characters in the Basic Multiingual Plane are mapped into a single 16 bit word, with other characters are mapped into a pair of 16 bit words.

UTF-32 is a fixed length encoding that uses 32 bits for every code point. This tends to make for a highly inefficient coding that is, generally, unnecessarily large, as most language uses of Unicode draw characters from the Basic Multilingual Plane, making the average code size 16 bits in UTF-16 as compared to the fixed length 32

bits in UTF-32. For this reason UTF-32 is far less common in terms of usage than UTF-8 and UTF-16.

UTF-8, UTF-16 and UTF-32 all require an "8-bit clean" storage and transmission medium. Unlike ASCII, which only uses seven bits per byte to encode characters, these UTF encodings need all eight bits per byte. Since domain names have been able to be represented with 7-bit ASCII characters, not all applications that process domain names preserve the status of the eighth bit in each byte; in other words, they are not 8-bit clean. This issue stimulated significant debate in the IETF's IDN Working Group and influenced the direction of the standards development into the area of application assistance by taking a very conservative view of the capabilities of the DNS as a restricted ASCII code application.

But languages, being something that we humans twist and torture in various ways every day, are not always entirely definitive in their use of characters, and Unicode has some weaknesses in terms of identifying a context of a script and a language for a given character sequence. The common approach to using Unicode encodings in application software is that of the use of an associated "tag", allowing content to be tagged with a script and an encoding scheme. For example, a content tag might read: "This text has been encoded using the KOI-8 encoding of the CRYILLIC script."

Cyrillic

The layout of this alphabet is derived from the early Cyrillic alphabet, itself a derivative of the Glagolitic alphabet, a ninth century uncial cursive usually credited to two Byzantine monk brothers from Thessaloniki, Saint Cyril and Saint Methodius.

Although it is widely accepted that the Glagolitic alphabet was invented by Saints Cyril and Methodius, the origins of the early Cyrillic alphabet are still a source of much controversy. Though it is usually attributed to Saint Clement of Ohrid, disciple of Saint Cyril and Saint Methodius from Bulgarian Macedonia, the alphabet is more likely to have developed at the Preslav Literary School in northeastern Bulgaria, where the oldest Cyrillic inscriptions have been found, dating back to the 940s. The theory is supported by the fact that the Cyrillic alphabet almost completely replaced the Glagolitic in northeastern Bulgaria as early as the end of the tenth century, whereas the Ohrid Literary School—where Saint Clement worked—continued to use the Glagolitic until the twelfth century. Of course, as the disciples of St. Cyril and Methodius spread throughout the First Bulgarian Empire, it is likely that these two main scholarly centers were a part of a single tradition.

Among the reasons for the replacement of the Glagolithic with the Cyrillic alphabet is the greater simplicity and ease of use of the latter and its closeness with the Greek alphabet, which had been well known in the First Bulgarian Empire.

[http://en.wikipedia.org/wiki/Cyrillic_alphabet]

Tagging allows the encoded characters to be decoded in the context of a given script and a given language. This has been useful for content, such as email content or web page content, but tagging breaks down in the context of the DNS. There is no natural space in DNS names to contain language and script tags. This implies that attempting to support internationalization in the DNS has to head towards a "universal" character set and a "universal" language context. Another way of looking at this is that the DNS must use an implicit tag of "all characters and all languages".

The contexts of use of DNS names have a number of additional artefacts. What about domain name label separators? You may need to map the '.' label separator, as '.' is an ASCII period character, and in some languages, such as Thai, there is no natural use of such a label separator. Also, are URLs intended to be end-user visible? If so then we may have to transform the punctuation components of the URL into the script of the language. Therefore, we may need to understand how to manage protocol strings, such as "http:" and separators such as the '/' character. To complete the integrity of the linguistic environment these elements may also require local presentation transformations.

> The Thai alphabet uses forty-four consonants and fifteen basic vowel characters. These are horizontally placed, left to right, with no intervening space, to form syllables, words, and sentences. Vowels associated with consonants are non-sequential: they can be located before, after, above or below their associated consonant, or in a combination of these positions. The latter in particular causes problems for computer encoding and text rendering.
>
> [http://www.omniglot.com/writing/thai.htm]

The DNS name string reads left to right, and not right to left or top to bottom as in other script and language cultures. How much of this you can encode in the DNS and how much must be managed by the application is part of the issue here. Is the effort to internationalize the DNS with multiple languages restricted to the "words" of the DNS, leaving the left-to-right ordering and the punctuation of the DNS unaltered? If so, how much of this is a poor compromise, in that the use of these DNS conventions in such languages not an entirely natural translation.

The DNS itself is a heavily restricted language of its own. The prudent use of the DNS specifies, in RFC 1035, a sequence of "words", where each word conforms to the "letter, digit, hyphen" restriction. The DNS "word" must begin with a letter (restricted to the Latin character subset of 'A' through 'Z' and 'a' through 'z'), followed by a sequence of letters, digits or hyphens, with a trailing letter or digit (no trailing hyphen). Furthermore, the case of the letter is not important to the DNS, so, within the DNS 'a' is equivalent to 'A', and so on, and all characters are encoded in monocase ASCII. The DNS uses a left-to-right ordering of these words, with the ASCII full stop as the word delimiter. This restriction is often referred to as the "LDH" convention. LDH is one of those acronyms that seem to proliferate when reading about the DNS. In this case "LDH" stands for "letter, digit, hyphen".

The challenge posed with the effort of *internationalizing* the DNS is one of attempting to create a framework that allows use of Internet applications, and the use of the DNS in particular, to be set in the user's own language in an entirely natural fashion, and yet allow the DNS to operate in a consistent and deterministic manner. In other words, we all should be able use browsers and email systems using our own language and scripts within the user interface, yet still be able to communicate naturally with others who may be using a different language interface.

The most direct way of stating the choice set of IDN design is that IDNs either change the "prudent use" of the deployed DNS into something quite different by permitting a richer character repertoire in all parts of the DNS, or IDNs change the applications that want to support a multi-lingual environment such that they have to perform some form of encoding transfer to map between a language string using Unicode characters and an "equivalent" string using the restricted DNS LDH character set repertoire. It appears that options other than these two lead us into fragmented DNS roots, and having already explored that space some years back not many of us are all that interested to return. So if we want to maintain a cohesive and unified symbol space for the DNS then either the deployed DNS has to clean up its act and become 8-bit clean, or applications have to do the work and present to the DNS an encoded form of the Unicode sequences that conform to the restricted DNS character repertoire.

The IDN Framework

For an English language user with the ASCII character set, the DNS name you enter into the browser, or the domain part of an email address, is almost the same string as the string that is passed to the DNS resolver to resolve into an address (the difference is the conversion of the characters into monocase). If you want to send a mail message you might send it to *user*@example.com, for example, and the domain name part of this address, example.com, is the string used to query the DNS for an MX Resource Record in order to establish how to actually deliver the message.

But what if I wanted to use a domain name that was expressed in another language? What if the email address was *user*@記念.com? The problem here is that this domain name cannot be 'naturally' expressed in the restricted syntax of the DNS, and while there may be a perfectly reasonable Unicode code sequence for this domain name, this encoded sequence is not a strict LDH sequence, nor is it case insensitive (for whatever "case" may mean in an arbitrary non-Latin script). It is here that IDNs take a step away from the traditional view of the DNS and use a hybrid approach to the task of mapping human strings into network addresses.

The Internationalized Domain Name (IDN) Working Group of the IETF was formed in 2000 with the goal of developing standards to internationalize domain names. The working group's charter was to specify a set of requirements and develop IETF standards-track protocols to allow a broader range of characters to be used in domain names. The outcome of this effort was the IDN in Applications (IDNA) framework, published as RFCs 3454, 3490, 3491, and 3492.

Rather than attempting to expand the character repertoire of the DNS itself, the IDN working group used the approach of using an *ASCII compatible encoding* (ACE), to encode the binary data of Unicode strings that would make up IDNs into an ASCII character encoding. The concept is very similar to the Base64 encoding used by the MIME email standards, but whereas Base64 uses 64 characters from ASCII, including uppercase and lowercase, the ACE approach requires the smaller DNS-constrained LDH subset of ASCII.

Various ACE algorithms were examined, and different algorithms have different compression goals (and yields) and encode the data using slightly different subsets of ASCII. Most proposals specified a prefix to the ACE coding to tag the fact that this was, in fact, an encoded Unicode string. One of the early ACE algorithms proposed was RACE (Row-based ASCII Compatible Encoding), and was widely implemented as a result of its use in VeriSign's IDN Testbed. To give an example, the RACE encoding of the domain name 記念*.com* in the early testbed was bq--3cfbqx7v.com. The two Chinese characters (記念) encode to 3cfbqx7v, and bq-- is the prefix indicating that particular label is encoded in RACE.

Since then the IETF adopted punycode as its standard IDN ACE (RFC 3492). Punycode was chosen for its encoding compression properties. Thus the domain name of記念*.com* (which encodes via RACE to bq--3cfbqx7v.com) encodes with punycode to xn--h7tw15g.com.

IDN in Applications (IDNA)

While an ASCII-compatible encoding of Unicode characters (an "ACE") allows an IDN to be represented in a form that will probably not be corrupted by the deployed DNS infrastructure on the Internet, an ACE alone is not a full solution. The IDN approach also needs to specify how and where the ACE should be applied.

The overall approach to IDNs is relatively straightforward (as long as you say it quickly enough and don't pause to consider it too deeply!). In IDN the application has a critical role to play. It is the responsibility of the application to take a domain name that is expressed in a particular language using a particular script, and potentially in a particular character and word order that is related to that language, and produce an ASCII-compatible encoded version of this DNS name, such that the ASCII-compatible encoding conforms to the DNS LDH restricted syntax. Equally, when presenting a DNS string to the user, the application should take the ASCII-compatible encoded DNS name and transform it to a presentation sequence of glyphs that correspond to the original string in the original script.

Although we'll see when looking at the *homoglyph* problem (a *homoglyph* is where two different code points share an identical glyph, such as the Greek language omicron 'o' (code point 03BF) and the ASCII lower case 'o' (code point 006F)) that the current general IDN approach in IDN-aware browsers is to deliberately not display the name as a Unicode code sequence using appropriate glyphs, but to display the punycode equivalent in ASCII text.

With IDNs the application now plays a critical role, and the importance of the application being capable of performing this encoding and decoding function correctly, deterministically and uniformly is critical to the entire IDN framework.

The basic shift in the DNS semantics that IDNs bring to the DNS is that the actual name itself is no longer in the DNS – an encoded version of the canonical name form sits in the DNS, and the users' applications have the responsibility to perform the canonical name transformation, as well as the mapping between the Unicode character string and the encoded DNS character string. So we have better all agree on what are the 'canonical' forms of name strings in each and every language, we'd also better all agree on this encoding, and our various applications had all had better run precise equivalents of these canonical name and encoding algorithms, or the symbolic consistency of the DNS is about to break. And that would be very messy indeed!

RFC3454 defines a *presentation layer* in IDN-aware applications that is responsible for the punycode ACE encoding and decoding. This new layer in the applications' architecture is be responsible for encoding any internationalized input in domain names into punycode format before the corresponding domain name is passed to the DNS for resolution. This presentation layer is also responsible for decoding the punycode format in IDNs and rendering the appropriate glyphs for the user.

It's all a case of your own personal perspective whether this is regarded as an elegant workaround, or whether you see this simply as a case of shifting an unresolved problem from one IETF Area to another one! The IDNA approach makes the assumption that it's easier to upgrade applications to all behave consistently in interpreting IDNs than it is to change the underlying DNS infrastructure to be 8-bit clean in a manner that would support direct use of Unicode code points in the DNS.

A slightly more cynical view is that its pretty apparent that the IETF's DNS folk have managed to outsource the hard parts of their problem to the IETF's applications folk on the basis that applications are relatively easy to change!

Interestingly enough, in another IETF area, the Internet Protocol folk are trying to grapple with the intricacies of the identifier/locator semantic overload of IP addresses, and has adopted the view that its extremely hard to alter application behaviour. These folk continue to examine ways to introduce more flexibility into the internet layer without exposing any parts of this change to applications.

The Presentation Layer Transform for IDNs

As a presentation layer transformation, the IDN framework allows names with non-ASCII characters for user input into a browser's location bar or URLs embedded in web pages, or as domain names in email addresses. At the DNS protocol level, there is no change in the restriction that only a subset of ASCII characters be used in the DNS protocol exchanges. If end users input non-ASCII characters as part of a domain name or if a web page contains a link using a non-ASCII domain name, the application must convert such potential DNS input into a special encoded format using only the usual ASCII subset characters.

The aim here is to define a reliable and deterministic algorithm that takes a Unicode string in a given language and produces a DNS string as expressed in the LDH character repertoire. This algorithm should not provide a unique 1:1 mapping, but should also group "equivalent" Unicode strings, where "equivalence is defined in the context of the language of use, into the same DNS LDH string. Any reverse mapping from the DNS LDH string into the Unicode string should deterministically select the single "canonical" string from the group of possible IDN strings.

Stringprep

The first part of the presentation layer transform is to take the original Unicode string and apply a number of transformations to it to produce a "regular" or "canonical" form of the IDN string. This is then transformed using the punycode ACE into a encoded DNS string form. The generic name of this process is termed, in IDN-language, "stringprep" (RFC3454) , and the particular profile of transformations used in IDNAs is termed "nameprep" (RFC3492).

This transform of a Unicode string into a canonical format is based on the observation that in a number of languages there are a variety of ways to display the same text and a variety of ways to enter the same text. While we humans have become entirely comfortable with this concept of multiple ways to express the same thing, the DNS is an exact equivalence match operation and it cannot tolerate any imprecision. So how can the DNS tell that two text strings are intended to be identical even though their Unicode strings are different when looked at as a sequence of 1's and 0's? The IDN approach is to transform the string so that all equivalent strings are mapped to the same canonical form, or "stringprep" the string. The stringprep specification is not a complete algorithm, and it requires a "profile" that describes the applicability of the profile, the character repertoire (at this time of writing RFC3454 it was Unicode 3.2, although the Unicode Consortium has subsequently released Unicode versions 4.0, 4.1 and 5.0), mapping tables normalization, and prohibited output characters.

Mapping

In converting from a string to a "normal" form, the first step is to map each character into its equivalent, using a mapping table. This is conventionally used to map characters to their lower case equivalent value to ensure that the DNS string comparison is case insensitive. Other characters are removed from the string via this mapping operation as their presence or absence in the string does not affect the outcome of a string equivalence operation, such as characters that affect glyph choice and placement, but without semantic meaning.

Mapping will create monocase (lower case to be specific) outcomes and also all eliminate non-significant code points, (such as, for example the code point 1806; MONGOLIAN TODO SOFT HYPHEN or code point 200B; ZERO WIDTH SPACE),

Normalization

A number of languages can use different character sequences for the same outcomes. Characters may appear the same in presentation format as a glyph sequence, yet have different underlying codepoints. This

may be associated with variables ways of combining diacritics, or using canonical codepoints, or using compatibility characters, and, in some language contexts, performing character reordering.

> For example, the character Ä can be represented by a single Unicode code point 00C4; LATIN CAPITAL LETTER A WITH DIAERESIS. Another valid representation of this character is the code point 0041; LATIN CAPITAL LETTER A followed by the separate code point 0398; COMBINING DIAERESIS.

The intent of normalization is to ensure that every class of character sequences that are equivalent in the context of a language are translated into a single canonical consistent format. This allows the equivalence operator to perform at the character level using direct comparison without language-dependent additional equivalence operations.

> On the other hand, Unicode normalization requires fairly large tables and somewhat complicated character reordering logic. The size and complexity should not be considered daunting except in the most restricted of environments, and needs to be weighed against the problems of user surprise from comparing unnormalized strings. [RFC3454]

In the real world of day-to-day language use languages are not rigid structures, and human use patterns of languages change. Normalization is no more than a best effort process to attempt to detect equivalences in a rigid rule-driven manner, and it may not always produce predictable outcomes. This can be a problem when regarded from the perspective of namespace collisions in the DNS, because it does not increase the confidence level of the DNS as a deterministic exact match information retrieval system. IDNs introduce some forms of name approximation into the DNS environment, and the DNS is extremely ill-suited to the related "fuzzy search" techniques that accompany such approximations.

Filtering Prohibited Characters

The last phase in string preparation is removing of prohibited characters. These include the various Unicode white space code points, control code points and joiners, private use code points and other code points used as surrogates or tags. A full enumeration of such characters can be found in Appendix C of RFC3454

Right-to-Left Characters

As an option for a particular stringprep profile, a check can be performed for right-to-left displayed characters, and if any are found, make sure that the whole string satisfies the requirements for bidirectional strings. The Unicode standard has an extensive discussion of how to reorder glyphs for display when dealing with bidirectional text such as Arabic or Hebrew. All Unicode text is stored in logical order as distinct to the display order.

Nameprep – a Stringprep profile for the DNS

The Nameprep profile specifies Stringprep for internationalized Domain Names, specifying a character repertoire (in this case the specification references Unicode 3.2), and a profile of mappings, normalization (form "KC"), prohibited characters and bidirectional character handling. The outcome of this is that two character sequences can be considered equivalent in the context of Internationalized Domain Names if, following this string prep profile, the resultant sequences of Unicode code points are identical. These are the "canonical" forms of names that are used by the DNS.

The Punycode ASCII-Compatible Encoding

The next step in the application's responsibility is to take this canonical form of the Unicode name string and transform it into a LDH equivalent string using an ACE. The algorithm used, punycode, is evidently an instance of a method termed "bootstring". The main attribute of this punycode algorithm is that the encoding is highly efficient, in that Unicode code point sequences do not become extended length ACE strings.

So, how does the algorithm work?

The punycode algorithm is described in RFC3492. The input code points are divided into a set of "basic" code points, the require no further encoding as they are already in the LDH character set, and the set of "extended" code points. The algorithm takes the basic code points and reproduces this sequence in the encoded string. This is the "literal portion" of the string. A delimiter is then added to the string. This delimiter is a basic code point that does not occur in the remainder of the string. The extended code points are then added to the string as a series of integers expressed through an encoding into the basic code set.

These additions of the extended code points are done primarily in the order of their Unicode-values, and secondarily in the order in which they occur in the string. The code for each insertion represents the number of possibilities of inserting a special character at the given stage (that is, without regard to characters that will be inserted afterwards), before the actual insertion, where these possible insertions are again ordered primarily according to their Unicode-values, and secondarily according to position. The encoding of the code point and its insertion position is done via a difference, or offset, encoding, so that sequences of clustered code points, such as would be found in a single language, encode very efficiently. In the case of multiple occurrences of a character it also helps that positions are counted from the previous position.

> For example, the German language string "mädchen", uses basic codes for all bar the 'ä' character. The punycode algorithm copies all the basic codes, followed by a '-'. The value and position of the ä insertion now has to follow, The encoded form for 'ä' (char 228) at the position between the first and second characters, This gives a delta code of 771. This value can be expressed as 22 x 35 + 1. 'b' corresponds to the value 1, and 'u' to the value 35. The code point and position information is represented in base 35 notation as (0,22,1), or, in reverse notation, with the encoding "bua". So the punycode encoding of "mädchen" is "mdchen-bua". The internationalized domain name format prepends the string "xn--" to the punycode string, resulting in the encoded domain name form of xn--mdchen-bua
>
> [http://en.wikipedia.org/wiki/IDNA]

IDNS and our assumptions about the DNS

At this stage it should be evident that we have the code points for characters drawn from all languages, and the means to create canonical forms of various words and express these in an encoded form that can be resolved via the DNS.

> IDNs are in use today of course. Here's a quick "sampler" of domain names expressing in a variety of languages:
>
> Arabic: عربي.com, ايكيا.com.
>
> Chinese: 宜家.com, 上海酒店.com,程序员.com.
>
> Greek: λλλ.com.
>
> Hebrew: שלום.com, ישראל.com.
>
> Hindi: खोज.com, भाषा.com.

Japanese: バドミントン.com, 瀬戸.net,江戸.jp.

Korean: 한글.kr, 현금영수증.kr.

Russian: доменные-имена.com, ИКЕА.com.
Spanish: viñadelmar.cl, ñandú.cl.
Symbols: ®.com, ©.com.

Traditional Chinese: 台灣大學.tw, 中大.tw.

Persian وب.سمپاد.ایران.ir

Tamil சினிமா.com

Thai เกมส์.com, ก.com

[http://en.wikipedia.org/wiki/Internationalized_domain_name]

Problem solved?

Unfortunately not!

While there are a massive number of discrete code points out there in Unicode-land, that does not imply that all these distinct characters are displayed in unique ways. Indeed given a relatively finite range of glyphs it should come as no surprise to learn that a number of discrete code points can be display with the same glyph.

The often-quoted example with IDNs and name confusion is the name paypal. What is the difference between www.paypal.com and www.paypal.com? There is a subtle difference in the first 'a' character, where the second domain name has replaced the Latin 'a' with the Cyrillic 'а'. Could you spot the difference? Of course not! These *homoglyphs* are cases where the underlying domain names are quite distinct, yet their screen appearance is indistinguishable. In the first case the domain name www.paypal.com is resolved in the DNS with the query string www.paypal.com, yet the second the query string www.paypal.com is translated by the application to the DNS query string www.xn--pypal-4ve.com. How can the user tell one case from the other?

The example given here is by no means a unique case in the IDN realm. The reports on "Unicode Security Considerations (Unicode Technical Report 36) and "Unicode Security Mechanisms" (Unicode Technical Report 39) provides many more examples of post normalization homographs. For example, the Tamil character 'ஸ' (Unicode code point 0BB5) and the Malayalam character 'ഩ' (Unicode code point 0D16) share a common glyph. Within the Tamil script the letter "Ta" 'க' (code 0BE7) and the digit "one" '௧' (code 0B95) are graphically identical.

There is no clear and unique relationship between characters and glyphs. Cyrillic, Latin and Greek share a number of common glyphs, glyphs may change their shape depending on the character sequence, multiple characters may produce a single glyph (such as the character pair "fl" being displayed as the single glyph 'ﬂ', and a single character may generate multiple glyphs.

Homoglyphs extend beyond a conventional set of characters and include syntax elements as well. For example, the Unicode point 0244 FRACTION SLASH is often displayed using the slash glyph, allowing URLs of the form "http://a.com/e.com". Despite its appearance this is not a reference to "a.com" with a locator suffix of "e.com", but is a reference to the domain "a.com/e.com"

The basic response is that if you maintain IDN integrity at the application level, then the user just can't tell. The punycode transform of www.paypal.com into www.xn--pypal-4ve.com is intended to be a secret between the application and the DNS, as this ASCII encoded form is simply meaningless to the user. But if this encoded form remains invisible to the user, how can the user detect that the two identically presented name strings are indeed different? Sadly, the only true 'security' we have in the DNS is the "look" of the DNS name that is presented to the user, and the user typically works on the principle that if the presented DNS string looks like the real thing, then it must be the real thing!

When this *homoglyph* issue was first exposed, the response from a number of browsers was to turn off all IDN support in their browser. The next response was to deliberately expose the punycode version of the URL in the browser's address bar, so that in both Explorer and Firefox, directing the browser to http://www.pаypal.com now displays in the address bar the URL value of http://www.xn--pypal-4ve.com. The distinction between the two names is now visible to the user, but the downside is that we are back to displaying ASCII names again, and in this case ASCII versions of punycode-encoded names. (If trying to "read" Base64 was hard, then the displaying, and understanding, of punycode is surely equally as hard, if not harder!) The encoded names can be completely devoid of any form of useful association or meaning. While the distinction between ASCII and Cyrillic may be evident by overt differences in their ASCII-encoded names, what is the case when the homoglyph occurs across two non-Latin languages? The punycode strings are different, but as to which string is the "intended" one? Did you mean http://xn--21bm4l.com or http://xn--q2buub.com when you entered the hindi script URL http://खोज.com? We appear to be back to guessing games in the DNS again, unfortunately, and particularly impossible guessing games at that. Displaying the URL in browsers in their punycode is an English language ASCII solution that detects *homoglyph* character substitution is being performed on what the user through was a 'plain' DNS ASCII string, but is of little value, if any, in non-ASCII contexts.

Now what was the objective of IDN support again? Was it to get around the use and display of ASCII names? We do not appear to be making a huge amount of progress here if we need to display the ASCII encoded forms as the only available means of prevention of various forms of passing-off and phishing attacks using the combination of DNS and IDNs. Wasn't the objective to create a DNS solution that was equally natural and equally 'secure' in the context of every language and every script? If the universal answer, when there is some element of doubt about the integrity of the DNS name, is to return to the base ASCII presentation of the punycode encoded DNS name then we've only succeeded in making the DNS more complex for non-English users, rather than easier and more 'natural'. Whoops.

If the intention in the IDN effort was to preserve the deterministic property of DNS resolution, such that a DNS query can be phrased deterministically and not have the query degenerate into a search term or require the application of fuzzy logic to complete the query, then we aren't quite there yet. The underlying observation is that languages are indeed human use systems. They can be tricky, and they invariable use what appear to be rules in strange and inconsistent ways. They are resistant to automated processing and the application of rigid rule sets. The canonical name forms that are produced by nameprep-like procedures are not comprehensive, nor does it appear that such a rigidly defined rule-driven system can produce the desired outcomes in all possible linguistic situations. And if the intention of the IDN effort was to create a completely "natural" environment using a language environment other than English and a display environment that is not reliant on ASCII and ASCII glyphs, while preserving all the other properties of the DNS, then the outcome does not appear to match our original IDN expectations.

The underlying weakness here is the implicit assumption that in the DNS what you see is what you get, and that two DNS names that look identical are indeed references to the same name and when resolved in the DNS produce precisely the same resolution outcome. When you broaden the repertoire of appearances of the DNS, such that the entire set of glyphs can be used in the DNS, then the mapping from glyph to underlying code point is not in and of itself unique. Any effort to undertake such a mapping needs additional context in the form of a language and script context. But the DNS does not carry such a context, making the task of maintaining uniqueness and determinism of DNS name translation essentially impossible if we also want to maintain the property that it's the appearance, or presentation format, of DNS names to the user that is the foundation stone of the integrity of our trust in the DNS.

IDNS, TLDs and the Politics of the DNS

So why is there a very active debate, particularly within ICANN-related forums, about putting IDN codes into the root of the DNS as alternative top level domains?

I have seen two major lines of argument here; namely the argument that favours the existence of IDNs in all parts of the DNS, including the top level domains, and the argument which favours a more restricted view of IDNs in the root of the DNS that links their use to that of an existing (ASCII-based) DNS label in the top level domain zone.

As far as I can tell, those who favour the approach of using IDNs in the top level zone as just another DNS label see this as a natural extension of adding punycode-encoded name entries into lower levels of the DNS. Why should the root of the DNS be any different, in terms of allowing IDNs? Why should a non-Latin script user of the Internet have to enter the top level domain code in its ASCII text form, while the remainder of the string is entered in a local language? And in right-to-left scripts, where does this awkward ASCII appendage sit when a user attempts to enter it into an application?

Surely, goes the argument, the more natural approach is to allow any DNS name to be wholly expressable in the language of the user, and this implies that all parts of the DNS should be able to carry native language-encoded DNS names. After all, コンピュータは予約する.jp looks just plain wrong as a mono-lingual domain name. What's that ".jp" appendage doing there in that DNS name? Surely a Japanese user should not have to resort to an ASCII English abbreviation to enter in the country code for Japan, when 日本 is obviously more 'natural' in the context of a Japanese use of Japanese script. If we had punycode top level domains then, goes the line of argument, a user could enter the entire domain name in their language and have the punycode encoding happen across the entire name string, and then successfully perform a DNS lookup on the punycode equivalent. This way the user would enter the Japanese character sequence: コンピュータは予約する.日本 and have the application translate this to the DNS string xn--88j0bve5g9bxg1ewerdw490b930f.xn--wgv71a. For this to work in its entirety in a uniform and consistent fashion the name xn--wgv71a needs to be a top level domain name.

> One can always take this one step further and take issue with the ASCII string http and the punctuation symbols :// for precisely the same reason, but I've not heard (yet) calls for multi-lingual equivalents of protocol identifier codes! The multi-lingual presentation of these elements remain firmly in the provenance of the application, rather than attempting to alter the protocol identifiers in the relevant standards.

The line of argument also encompasses the implicit threat that if the root of the DNS does not embrace top level domains as expressed in the language of the Internet's users, then language communities will break away from a single DNS root and meet their linguistic community's requirements in their own DNS hierarchy. Citing the recent case of the Chinese language emulated top level domain equivalent of .com (the chinese-language equivalent is expressed phonetically as .gongsi), the argument expresses the view that admitting such encoded tags into the DNS root is the lesser of many potential evils, including the consequence of inactivity, which is cited as being tantamount to condoning the complete fragmentation of the Internet's symbol set.

> To set the record straight, the Chinese effort at creating Chinese language equivalents of some of the generic top level domain names did not rely on split DNS roots and deliberate fragmentation of the DNS. The exercise used an application-level plugin that appended the ASCII string .cn to the domain names before passing then to the DNS for resolution, but chose not to display this ASCII appendage at the user level. This approach did not 'break' or 'fracture' the DNS in

any way, and was more or less a recycling of application plug-in technology that used internal top level name synthesis that first generally appeared in the closing stages of the Internet boom in 2000 – 2001. The willingness of many observers to ascribe this apparent exercise in fracturing the DNS root to a deliberately provocative national agenda is illustrative of the consistently escalating levels of tension that the entire DNS topic has managed to engender over the past decade.

Its not easy to disentangle a number of threads of difference of perspective here, not the least of which is a continuing sense of frustration in many parts of the world over what they perceive as the continuing undue level of influence of US-based enterprises and the US government over the Internet, its applications and even the underlying technical standards. The desire to find alternatives to .com and their generic top level domain label cohorts may well rest as much in the desire to create more diversified operators of these top level domain names that are well-distanced from US commercial and governmental interests as they are based in a desire to escape from the linguistic imperialism of ASCII names. Of course such motives also sit beside the observation that in the domain name registration business the typical registry charge of some $US 6.00 per domain name year is significantly higher than the operational costs that appear generally to be well under $US 2.00 per domain name year. Little wonder, therefore, that there could be the desire to create a language based "monopoly" name retailer that supplants the generic ASCII generic TLDs in a particularly populous language-use locale.

Of course having an entirely new top level domain name in an IDN name format does not solve the entirety of the potential issues with IDNs. How can a user tell what domain names are in the ASCII top level, and what are in the "equivalent" IDN encoded top level domains? Are any two name spaces, which refer to the same underlying name concept, equivalent. Is xn--88j0bve5g9bxg1ewerdw490b930f appropriately a subdomain of .jp, or a subdomain of xn--wgv71a? Should the two domains be tightly synchronized with respect to their zone content and represent the same underlying token set, or should they be independent offerings to the market place, and allow registrants and the end user base make implicit choices here? In other words, should the pair of domain names, namely xn--88j0bve5g9bxg1ewerdw490b930f.xn--wgv71a and xn--88j0bve5g9bxg1ewerdw490b930f.jp, reference precisely the same DNS zone, or should they be allowed to compete, and each find their own 'natural' level of market support based on decoupled top level domain names of .jp and .xn--wgv71a?

What does the term *equivalence* really imply here? Is equivalence something as loose as the relationship between .com and .biz, namely being different abbreviations of words that reflect similar concepts with different name space populations that reflect market diversity and a competitive supply industry. Or is equivalence a much tighter binding in that equivalent names share precisely the same sub-domain name set, and that a registration in one of these equivalence names is in effect a name registration across the entire equivalence set?

Even this is not a readily resolveable issue given our various interpretations of equivalence. In theory, the DNS root zone is populated by ISO 2-letter country codes and a number of "generic" top level domains. Under what basis, and under what authority, is xn--wgv71a considered an "equivalent" of the ISO 3166 two letter country code "JP". Are we falling into the trap once again of making up the rules as we go along here? Is the distinction between "com" and "biz" only one that is apparent in English? And why should this apply only to non-latin character sets? Surely it makes more sense for a native German language speaker to refer to commercial entities as *kommerze*, and the abbreviated TLD name as .kom?

Equivalence in a linguistic sense is a verv tough topic. For a presentation I was wanting to use the opposite of the phrase "*Let 100 flowers bloom; let a hundred schools of thought contend*", or in Chinese script, "百花齊放，百家爭鳴". In English the opposite of this thought is easy to express: "*Let one flower bloom; let one school of thought prevail*". It appears that a tolerably close Chinese script equivalent is "一花独放，一家主鸣". But why is this not an exact equivalence?

I consulted my friend Mark Williams for assistance in finding an equivalent Chinese text that was the opposite of Mao's 100 flowers saying. When Mark is not travelling he lives in Beijing, and he is a keen student of the Chinese language. I thought that I was asking for a simple translation, but as it turned out I really did not understand the task of the language translator at all well! Chinese is a venerable language, and including all or part of traditional sayings into one's writings or speech is an integral part of Chinese language use. In English-speaking cultures we often refer to such devices as aphorisms, which has slightly disparaging overtones, but not so in Chinese. Mao cleverly constructed his phrase by putting parts of two sayings together, leaving the couplet of four character constructs in place, but adding through the juxtaposition of two different thoughts, his own touch.

To undertake the translation in a faithful manner Mark came up with a similar construct. The first four characters, "*Let one flower bloom (only one flower is allowed to bloom)*" comes from a common Chinese saying, in the same style of Mao's saying. The second part Mark had to construct in the style of a saying. "*One house (school of thought) alone be heard*" is formed again using four characters.

The lesson for me was that translation and equivalence are not just issues with single words, but it's the style and context of the text that really create the sense of a "natural" equivalence across languages. And when searching for language equivalence across languages that do not share a common linguistic root, the task is far more challenging. In this case I had asked for a translation of a linguistic artifice based on a "*poetic proverb*". A phrase that not just had meaning but a cadence and a tone. The equivalent expression, to make sense, also needed to reproduce the same style. The DNS is of, of course, incapable of expressing such concepts of linguistic style when considering issues of equivalence and canonical name forms.

I have a new respect for those who embark on the course of learning Chinese. This exercise has, for me, been for me a fascinating education in the deeper aspects of symbols and their use in cultures that thread through millennia.

Lets put aside the somewhat difficult concept of name equivalence for a second, and just make the breathtaking assumption that this entire equivalence problem is a solved problem. Lets also suppose that we want tight coupling across equivalence sets of names.

In other words, what we would like is that a name registered in any of the elements of the equivalent domain name set is, in effect, registered in all the equivalent DNS zones. The question is: How should it be implemented in the DNS? One approach that could support of tight synchronization of equivalence is the use the DNAME record (RFC 2672) to create these top level domain name aliases for their ASCII equivalents, thereby allowing a single name registration to be resolveable using a root name expressed in any of the linguistic equivalents of the original tld name. The DNAME entry for all but the "canonical" element of the equivalence set effectively translates all queries to a query on the canonical name. The positive aspects of such an approach is uniformity across linguistic equivalents of the tld name form – a single name delegation in a tld domain becomes a name within all the linguistic equivalents of the tld name without any further delegation or registration required.

Of course DNAME itself is not entirely straightforward, given that if the name resolver indicates (via EDNS0) that it understands DNAME record, then the name server can return the DNAME record and the resolver will continue the query with the new name. If the resolver does not understand the DNAME records the server has to synthesize a CNAME response that will redirect the resolver to make a new query for this name. One difference lies in the cache properties. A DNAME-aware resolver will cache the DNAME response for the entire mapped name space, while a CNAME-aware resolver will only cache individual responses, and related queries will each be passed onto the server. A more critical difference lies in the face that the server now has an additional activity that increases the server load. DNAME would be an entirely neutral option, from the perspective of the server, were it not for this CNAME synthesis. So in looking at DNAMEs in the root, the here and now is not good: one of the root server implementations, NSD, does not support DNAME, there is not a large body of experience in the issues relating to CNAME synthesis, and the IETF, in the guise of the DNSEXT working group, appears to be entertaining thoughts about redefining DNAME records (draft-ietf-dnsext-rfc2672bis-dname-00.txt). Changing DNS RR types to reflect this name equivalence behaviour without the CNAME overtones is on the cards, but it would not quite be DNAME as we understand it today!

So, its still early days for DNAME as a tool to support sets of equivalent names in the DNS. The limited experience so far with DNAME indicates that CNAME synthesis places load back on the name servers that would otherwise not be there, and the combination of this synthetic record and DNSSEC starts to get very unwieldy. Also, the IETF is reviewing the DNAME spec with an eye to removing the requirement to perform CNAME synthesis. All of which may explain why there is no immediate desire to place DNAMEs in the DNS root zone.

Different interpretations are possible. The use of DNAMEs as aliases for existing top level domains in effect "locks up" IDNs into the hands of the incumbent tld registry operators . Part of the IDN debate, is, as usual, a debate over the generic TLDs registry operators and the associated perception of well–entrenched monopoly-based enterprises. Without DNAMEs it is feasible to allow multiple registrars with different IDN variants of the same gTLD. From the perspective of a coherent symbol space where the same symbol, expressed in any language script, resolves in the same fashion, then independent registries are not overly consistent with such a model of registry diversity in a multi-lingual environment.

It appears that another line of argument is along the lines that the DNS top level name space is one that is very carefully managed, and new entries into this space are not made lightly. There are issues of stability of operation, of attempting to conserve a coherent namespace, and the ever present consideration that if we manage to 'break' the DNS root zone it may be too late to recover. This line of argument recognizes the very hazy nature of name equivalence in a multi-lingual environment and is based on the proposition that the DNS is incapable of representing such imprecision with any utility. The DNS is not a search engine, and Verisign's thwarted efforts of Sitefinder in the past simply underscore the fact that the DNS does not handle imprecision well. Again, goes the argument, if this is the case then can we push this problem back to the application rather than trying to bend the DNS? If an application is capable of translating, say, 日本 into xn--wgv71a, and considering that the top level domain name space is relatively small, it appears that having the application performing a further translation of this intermediate form punycode string into the ASCII string jp is not a particularly challenging form of table lookup. In such a model no tld aliases or equivalences are required in the root zone of the DNS. If we are prepared to pass the execution of the presentation layer of the DNS to the application layer to perform, then why not also ask this same presentation layer to perform the step of further mapping the punycode equivalents of the existing top level domains to the actual top level domains, using some richer language context that the application may be aware of that is not viable strictly within the confines of the DNS?

So, with respect to the question of whether IDN TLDS should be loaded into the DNS at all, and, if so, whether they should represent an opportunity for further diversity in name supply or be constrained to be aligned to existing names, and precisely now name equivalence is to be interpreted in this context, then it appears that ICANN has managed to wedge itself in a challenging situation, as usual. In not making a decision those with an interest in having diverse IDN TLDs appear to derive some pleasure in pointing out that the political origins of ICANN and its strong linguistic bias to English are influencing it to ignore non-English language use and non-English language users of the Internet, and where dramatic statements are called for use terms such as "cultural imperialism" to illustrate the nature of the linguistic insult! The case has been made repeatedly, in support of IDN TLDs, that overwhelming majority of Internet users and commercial activity of the Internet is in languages other than native English, and the imposition of ASCII labels on the DNS is an unnatural imposition on these overwhelmingly majority of users.

On the other hand, most decisions to permit some form of entry in the DNS are generally seen an irrevocable step, and building a DNS that is littered with the legacy of various non-enduring name technologies and poor ad hoc decisions to address a particular issue or problem without any context of a longer term framework seems also to represent a step along a direction leading to a heavily fragmented Internet where users cannot communicate with each other.

Talking about IDNs has undoubtedly raised the political tenor of the debate. We are now considering matters that directly relate to topics that are of national significance, and national governments now have a entirely reasonable rationale for entering the IDN debate. There are a set of national agendas at play here, and part of the issue is that the relatively piecemeal use, and poor handling of scripts in the available set of application plug-ins for IDNs further add to the pressure to fragment the single DNS hierarchy into multiple distinct roots. It would be a shame if these various efforts, that can be tagged as various forms of localization triumph over the desire to have a single international network that provides a single and consistent communications medium to each and every one of us.

What about global interoperability and the Internet? Should we just take the easy answer and simply give up on the entire concept? Well of course not! But, taking a narrower perspective, are IDNs simply not viable in the DNS? I'd suggest that not only is this question one that has been overtaken by events years ago, but even if we want to reconsider if now, then the answer remains that any user using their local language and local script should have an equally 'natural' experience. IDNs are a necessary and valuable component of the symbol space of any global communications system, and the Internet is no exception here. However, in saying that we also should recognize that in saying that we really do need combinations of both localization and globalization, and that we are voicing some pretty tough objectives. Is it really the case that the IDNA approach is enough? Are our assumptions that an unaltered DNS with application-encoded name strings really a rich enough platform to preserve the essential properties of the DNS while allowing true multi-lingual use of the DNS? On the other hand, taking a pragmatic view of the topic, is what we have with IDNA enough for us to work on, and the alternative of re-engineering the entire fabric of the DNS just not a viable option?

I suspect that the framework of IDNA is now the technology for IDNs for the Internet, and we simply have to move on from here and deliberately take the stance of understanding the space from users' perspectives when we look at the policy issues of IDNs. The salient questions from such perspectives are: "What's the 'natural' thing to do?" and "What causes a user the least amount of surprise?". Because in this world it's definitely the case that what works for the user is what works for the Internet as a whole.

Further Reading

Its possible to reference an overwhelming amount of commentary on this topic, so I've deliberately kept this list relatively brief:

- Internationalizing Top-Level Domain Names: Another Look, John Klensin, ISOC Member Briefing, September 2004
  http://www.isoc.org/briefings/018/

- "National and Local Characters for DNS Top Level Domain (TLD) Names", John Klensin, October 2005
  RFC4185

- Papers submitted to the ICANN IDN TLD workshop, held in November 2005
  http://www.icann.org/announcements/announcement-17nov05.htm

- "Review and Recommendations for Internationalized Domain Names (IDNs)", Internet Architecture Board, September 2006
  RFC4690

- ICANN's IDN Roadmap Announcement - Progress and Future, ICANN,

http://www.icann.org/announcements/announcement-1-01nov06.htm

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre, nor those of the Internet Society.

About the Author

GEOFF HUSTON holds a B.Sc. and a M.Sc. from the Australian National University. He has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. He is author of a number of Internet-related books, and is currently the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He was a member of the Internet Architecture Board from 1999 until 2005, and served on the Board of the Internet Society from 1992 until 2001.

http://www.potaroo.net