Internet Draft
draft-ietf-idn-lace-00.txt
November 6, 2000
Expires May 6, 2001

Mark Davis
IBM
Paul Hoffman
IMC & VPNC

LACE: Length-based ASCII Compatible Encoding for IDN

Status of this memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

> The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt
>
> The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

This document describes a transformation method for representing non-ASCII characters in host name parts in a fashion that is completely compatible with the current DNS. It is a potential candidate for an ASCII-Compatible Encoding (ACE) for internationalized host names, as described in the comparison document from the IETF IDN Working Group. This method is based on the observation that many internationalized host name parts will have a few substrings from a small number of rows of the ISO 10646 repertoire. Run-length encoding for these types of host names will be fairly compact, and is fairly easy to describe.

1. Introduction

There is a strong world-wide desire to use characters other than plain ASCII in host names. Host names have become the equivalent of business or product names for many services on the Internet, so there is a need to make them usable by people whose native scripts are not representable by ASCII. The requirements for internationalizing host names are described in the IDN WG's requirements document, [IDNReq].

The IDN WG's comparison document [IDNComp] describes three potential

main architectures for IDN: arch-1 (just send binary), arch-2 (send binary or ACE), and arch-3 (just send ACE). LACE is an ACE, called Row-based ACE or LACE, that can be used with protocols that match arch-2 or arch-3. LACE specifies an ACE format as specified in ace-1 in [IDNComp]. Further, it specifies an identifying mechanism for ace-2 in [IDNComp], namely ace-2.1.1 (add hopefully-unique legal tag to the beginning of the name part).

In formal terms, LACE describes a character encoding scheme of the ISO/IEC 10646 [ISO10646] coded character set (whose assignment of characters is synchronized with Unicode [Unicode3]) and the rules for using that scheme in the DNS. As such, it could also be called a "charset" as defined in [IDNReq].

The LACE protocol has the following features:

- There is exactly one way to convert internationalized host parts to and from LACE parts. Host name part uniqueness is preserved.

- Host parts that have no international characters are not changed.

- Names using LACE can include more internationalized characters than with other ACE protocols that have been suggested to date. LACE-encoded names are variable length, depending on the number of transitions between rows in the ISO 10646 repertoire that appear in the name part. Name parts that cannot be compressed using run-length encoding can have up to 17 characters, and names that can be compressed can have up to 35 characters. Further, a name that has just a few row transitions typically can have over 30 characters.

It is important to note that the following sections contain many normative statements with "MUST" and "MUST NOT". Any implementation that does not follow these statements exactly is likely to cause damage to the Internet by creating non-unique representations of host names.

1.1 Terminology

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

Hexadecimal values are shown preceded with an "0x". For example, "0xa1b5" indicates two octets, 0xa1 followed by 0xb5. Binary values are shown preceded with an "0b". For example, a nine-bit value might be shown as "0b101101111".

Examples in this document use the notation from the Unicode Standard [Unicode3] as well as the ISO 10646 names. For example, the letter "a" may be represented as either "U+0061" or "LATIN SMALL LETTER A".

LACE converts strings with internationalized characters into strings of US-ASCII that are acceptable as host name parts in current

DNS host naming usage. The former are called "pre-converted" and the latter are called "post-converted".

1.2 IDN summary

Using the terminology in [IDNComp], LACE specifies an ACE format as specified in ace-1. Further, it specifies an identifying mechanism for ace-2, namely ace-2.1.1 (add hopefully-unique legal tag to the beginning of the name part).

LACE has the following length characteristics. In this list, "row" means a row from ISO 10646.

- LACE-encoded names are variable length, depending on the number of transitions between rows that appear in the name part.

- Name parts that cannot be compressed using run-length encoding can have up to 17 characters.

- Names that can be compressed can have up to 35 characters.

-A name that has just a few row transitions typically can have over 30 characters.

2. Host Part Transformation

According to [STD13], host parts must be case-insensitive, start and end with a letter or digit, and contain only letters, digits, and the hyphen character ("-"). This, of course, excludes any internationalized characters, as well as many other characters in the ASCII character repertoire. Further, domain name parts must be 63 octets or shorter in length.

2.1 Name tagging

All post-converted name parts that contain internationalized characters begin with the string "bq--". (Of course, because host name parts are case-insensitive, this might also be represented as "Bq--" or "bQ--" or "BQ--".) The string "bq--" was chosen because it is extremely unlikely to exist in host parts before this specification was produced. As a historical note, in late August 2000, none of the second-level host name parts in any of the .com, .edu, .net, and .org top-level domains began with "bq--"; there are many tens of thousands of other strings of three characters followed by a hyphen that have this property and could be used instead. The string "bq--" will change to other strings with the same properties in future versions of this draft.

Note that a zone administrator might still choose to use "bq--" at the beginning of a host name part even if that part does not contain internationalized characters. Zone administrators SHOULD NOT create host part names that begin with "bq--" unless those names are post-converted

names. Creating host part names that begin with "bq--" but that are not post-converted names may cause two distinct problems. Some display systems, after converting the post-converted name part back to an internationalized name part, might display the name parts in a possibly-confusing fashion to users. More seriously, some resolvers, after converting the post-converted name part back to an internationalized name part, might reject the host name if it contains illegal characters.

2.2 Converting an internationalized name to an ACE name part

To convert a string of internationalized characters into an ACE name part, the following steps MUST be preformed in the exact order of the subsections given here.

If a name part consists exclusively of characters that conform to the host name requirements in [STD13], the name MUST NOT be converted to LACE. That is, a name part that can be represented without LACE MUST NOT be encoded using LACE. This absolute requirement prevents there from being two different encodings for a single DNS host name.

If any checking for prohibited name parts (such as ones that are prohibited characters, case-folding, or canonicalization) is to be done, it MUST be done before doing the conversion to an ACE name part.

The input name string consists of characters from the ISO 10646 character set in big-endian UTF-16 encoding. This is the pre-converted string.

Characters outside the first plane of characters (those with codepoints above U+FFFF) MUST be represented using surrogates, as described in the UTF-16 description in ISO 10646.

2.2.1 Compress the pre-converted string

The entire pre-converted string MUST be compressed using the compression algorithm specified in section 2.4. The result of this step is the compressed string.

2.2.2 Check the length of the compressed string

The compressed string MUST be 36 octets or shorter. If the compressed string is 37 octets or longer, the conversion MUST stop with an error.

2.2.3 Encode the compressed string with Base32

The compressed string MUST be converted using the Base32 encoding described in section 2.5. The result of this step is the encoded string.

2.2.4 Prepend "bq--" to the encoded string and finish

Prepend the characters "bq--" to the encoded string. This is the host

name part that can be used in DNS resolution.

2.3 Converting a host name part to an internationalized name

The input string for conversion is a valid host name part. Note that if any checking for prohibited name parts (such as prohibited characters, case-folding, or canonicalization is to be done, it MUST be done after doing the conversion from an ACE name part.

If a decoded name part consists exclusively of characters that conform to the host name requirements in [STD13], the conversion from LACE MUST fail. Because a name part that can be represented without LACE MUST NOT be encoded using LACE, the decoding process MUST check for name parts that consists exclusively of characters that conform to the host name requirements in [STD13] and, if such a name part is found, MUST beconsidered an error (and possibly a security violation).

2.3.1 Strip the "bq--"

The input string MUST begin with the characters "bq--". If it does not, the conversion MUST stop with an error. Otherwise, remove the characters "bq--" from the input string. The result of this step is the stripped string.

2.3.2 Decode the stripped string with Base32

The entire stripped string MUST be checked to see if it is valid Base32 output. The entire stripped string MUST be changed to all lower-case letters and digits. If any resulting characters are not in Table 1, the conversion MUST stop with an error; the input string is the post-converted string. Otherwise, the entire resulting string MUST be converted to a binary format using the Base32 decoding described in section 2.5. The result of this step is the decoded string.

2.3.3 Decompress the decoded string

The entire decoded string MUST be converted to ISO 10646 characters using the decompression algorithm described in section 2.4. The result of this is the internationalized string.

2.4 Compression algorithm

The basic method for compression is to reduce a substring that consists of characters all from a single row of the ISO 10646 repertoire to a count octet followed by the row header followed by the lower octets of the characters. If this ends up being longer than the input, the string is not compressed, but instead has a unique one-octet header attached.

Although the uncompressed mode limits the number of characters in a LACE name part to 17, this is still generally enough for almost all names in almost scripts. Also, this limit is close to the limits set by other encoding proposals.

Note that the compression and decompression rules MUST be followed exactly. This requirement prevents a single host name part from having two encodings. Thus, for any input to the algorithm, there is only one possible output. An implementation cannot chose to use one-octet mode or two-octet mode using anything other than the logic given in this section.

2.4.1 Compressing a string

The input string is in big-endian UTF-16 encoding with no byte order mark.

Design note: No checking is done on the input to this algorithm. It is assumed that all checking for valid ISO/IEC 10646 characters has already been done by a previous step in the conversion process.

1) If the length of the input is not even, or is less than 2, stop with an error.

2) Set the input pointer, called IP, to the first octet of the input string.

3) Set the variable called HIGH to the octet at IP.

4) Determine the number of pairs at or after IP that have HIGH as the first octet; call this COUNT.

5) Put into an output buffer the single octet for COUNT followed by the single octet for HIGH, followed by all those low octets. Move IP to the end of those pairs; that is, set IP to IP+(2*(COUNT+1)).

6) If IP is not at the end of the input string, go to step 3.

7) If the length of the output buffer is less than or equal to the length of the input buffer (in octets, not in characters), output the buffer. Otherwise, output the octet 0xFF followed by the input buffer. Note that there can only be one possible representation for a name part, so that outputting the wrong name part is a serious security error. Decompression schemes MUST accept only the valid form and MUST NOT accept invalid forms.

2.4.2 Decompressing a string

1. Set the input pointer, called IP, to the first octet of the input string. If there is no first octet, stop with an error.

2. If the octet at IP is 0xFF, go to step 10.

3. Get the octet at IP, call it COUNT. Set IP to IP+1. If IP is now at the end of the input string, stop with an error.

4. Get the octet at IP, call it HIGH. Set IP to IP+1. If IP is now at the end of the input string, stop with an error.

5. Get the octet at IP, call it LOW. Set IP to IP+1.

6. Output HIGH, then LOW, to the output buffer.

7. Decrement COUNT. If COUNT is greater than 0, go to step 5.

8. If IP is not at the end of the input buffer, go to step 3.

9. Compare the length of the input string with the length of the output buffer. If the length of the output buffer is longer than the length of the input buffer, stop with an error because the wrong compression form was used. Otherwise, send out the output buffer and stop.

10. Set IP to IP+1. Copy the rest of the input buffer to the output buffer. Compress the output buffer into a separate comparison buffer following the steps for compression above. If the length of the comparison buffer is less than or equal to the length of the output buffer, stop with an error because the wrong compression form was used. Otherwise, send out the output buffer and stop.

2.4.3 Compression examples

The five input characters <U+30E6 U+30CB U+30B3 U+30FC U+30C9> are represented in big-endian UTF-16 as the ten octets <30 E6 30 CB 30 B3 30 FC 30 C9>. All the code units are in the same row (03). The output buffer has seven octets <05 30 E6 CB B3 FC C9>, which is shorter than the input string. Thus the output is <05 30 E6 CB B3 FC C9>.

The four input characters <U+012E U+0110 U+014A U+00C5> are represented in big-endian UTF-16 as the eight octets <01 2E 01 10 01 4A 00 C5>. The output buffer has eight octets <03 01 2E 10 4A 01 00 C5>, which is the same length as the input string. Thus, the output is <03 01 2E 10 4A 01 **00 C5>.**

The three input characters <U+012E U+00D0 U+014A> are represented in big-endian UTF-16 as the six octets <01 2E 00 D0 01 4A>. The output buffer is nine octets <01 01 2E 01 00 D0 01 01 4A>, which is longer than the input buffer. Thus, the output is <FF 01 2E 00 D0 01 4A>.

2.5 Base32

In order to encode non-ASCII characters in DNS-compatible host name parts, they must be converted into legal characters. This is done with Base32 encoding, described here.

Table 1 shows the mapping between input bits and output characters in Base32. Design note: the digits used in Base32 are "2" through "7" instead of "0" through "6" in order to avoid digits "0" and "1". This

helps reduce errors for users who are entering a Base32 stream and may misinterpret a "0" for an "O" or a "1" for an "l".

Table 1: Base32 conversion

bits	char	hex	bits	char	hex
00000	a	0x61	10000	q	0x71
00001	b	0x62	10001	r	0x72
00010	c	0x63	10010	s	0x73
00011	d	0x64	10011	t	0x74
00100	e	0x65	10100	u	0x75
00101	f	0x66	10101	v	0x76
00110	g	0x67	10110	w	0x77
00111	h	0x68	10111	x	0x78
01000	i	0x69	11000	y	0x79
01001	j	0x6a	11001	z	0x7a
01010	k	0x6b	11010	2	0x32
01011	l	0x6c	11011	3	0x33
01100	m	0x6d	11100	4	0x34
01101	n	0x6e	11101	5	0x35
01110	o	0x6f	11110	6	0x36
01111	p	0x70	11111	7	0x37

2.5.1 Encoding octets as Base32

The input is a stream of octets. However, the octets are then treated as a stream of bits.

Design note: The assumption that the input is a stream of octets (instead of a stream of bits) was made so that no padding was needed. If you are reusing this algorithm for a stream of bits, you must add a padding mechanism in order to differentiate different lengths of input.

1) Set the read pointer to the beginning of the input bit stream.

2) Look at the five bits after the read pointer. If there are not five bits, go to step 5.

3) Look up the value of the set of five bits in the bits column of Table 1, and output the character from the char column (whose hex value is in the hex column).

4) Move the read pointer five bits forward. If the read pointer is at the end of the input bit stream (that is, there are no more bits in the input), stop. Otherwise, go to step 2.

5) Pad the bits seen until there are five bits.

6) Look up the value of the set of five bits in the bits column of Table 1, and output the character from the char column (whose hex value is in the hex column).

2.5.2 Decoding Base32 as octets

The input is octets in network byte order. The input octets MUST be values from the second column in Table 1.

1) Set the read pointer to the beginning of the input octet stream.

2) Look up the character value of the octet in the char column (or hex value in hex column) of Table 1, and output the five bits from the bits column.

3) Move the read pointer one octet forward. If the read pointer is at the end of the input octet stream (that is, there are no more octets in the input), stop. Otherwise, go to step 2.

2.5.3 Base32 example

Assume you want to encode the value 0x3a270f93. The bit string is:

```
3   a    2   7    0   f    9   3
00111010 00100111 00001111 10010011
```

Broken into chunks of five bits, this is:

```
00111 01000 10011 10000 11111 00100 11
```

Padding is added to make the last chunk five bits:

```
00111 01000 10011 10000 11111 00100 11000
```

The output of encoding is:

```
00111 01000 10011 10000 11111 00100 11000
  h     i     t     q     7     e     y
```

or "hitq7ey".

3. Security Considerations

Much of the security of the Internet relies on the DNS. Thus, any change to the characteristics of the DNS can change the security of much of the Internet. Thus, LACE makes no changes to the DNS itself.

Host names are used by users to connect to Internet servers. The security of the Internet would be compromised if a user entering a single internationalized name could be connected to different servers based on different interpretations of the internationalized host name.

LACE is designed so that every internationalized host name part can be represented as one and only one DNS-compatible string. If there is any way to follow the steps in this document and get two or more

different results, it is a severe and fatal error in the protocol.

4. References

[IDNComp] Paul Hoffman, "Comparison of Internationalized Domain Name Proposals", draft-ietf-idn-compare.

[IDNReq] James Seng, "Requirements of Internationalized Domain Names", draft-ietf-idn-requirement.

[ISO10646] ISO/IEC 10646-1:1993. International Standard -- Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane. Five amendments and a technical corrigendum have been published up to now. UTF-16 is described in Annex Q, published as Amendment 1. 17 other amendments are currently at various stages of standardization. [[[THIS REFERENCE NEEDS TO BE UPDATED AFTER DETERMINING ACCEPTABLE WORDING]]]

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

[STD13] Paul Mockapetris, "Domain names - implementation and specification", November 1987, STD 13 (RFC 1035).

[Unicode3] The Unicode Consortium, "The Unicode Standard -- Version 3.0", ISBN 0-201-61633-5. Described at <http://www.unicode.org/unicode/standard/versions/Unicode3.0.html>.

A. Acknowledgements

Base32 is quite obviously inspired by the tried-and-true Base64 Content-Transfer-Encoding from MIME.

B. IANA Considerations

There are no IANA considerations in this document.

C. Author Contact Information

Mark Davis
IBM
10275 N. De Anza Blvd
Cupertino, CA 95014
mark.davis@us.ibm.com and mark.davis@macchiato.com

Paul Hoffman
Internet Mail Consortium and VPN Consortium

127 Segre Place
Santa Cruz, CA 95060 USA
paul.hoffman@imc.org and paul.hoffman@vpnc.org