Internet Draft
draft-ietf-idn-race-03.txt
November 22, 2000
Expires in six months

Paul Hoffman
IMC & VPNC

RACE: Row-based ASCII Compatible Encoding for IDN

Status of this memo

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 all their characters in one row of the ISO 10646 **repertoire.**

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). RACE is an ACE, called

Row-based ACE or RACE, that can be used with protocols that match arch-2 or arch-3. RACE 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).

Author's note: although earlier drafts of this document supported the ideas in arch-3, I no longer support that idea and instead only support arch-2. Of course, someone else might right an IDN proposal that matches arch-3 and use RACE as the protocol.

In formal terms, RACE 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 RACE protocol has the following features:

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

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

- Names using RACE can include more internationalized characters than with other ACE protocols that have been suggested to date. Names in the Han, Yi, Hangul syllables, or Ethiopic scripts can have up to 17 characters, and names in most other scripts can have up to 35 characters. Further, a name that consist of characters from one non-Latin script but also contains some Latin characters such as digits or hyphens can have close to 33 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".

RACE 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], RACE 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).

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

- If the characters in the input all come from the same row, up to 35 characters per name part are allowed.

- If the characters in the input come from two or more rows, neither of which is row 0, up to 17 characters per name part are allowed.

- If the characters in the input come from two rows, one of which is row 0, between 17 and 33 characters per name part are allowed.

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.

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.

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.

2.2.1 Check the input string for disallowed names

If the input string consists only of characters that conform to the host name requirements in [STD13], the conversion MUST stop with an error.

2.2.2 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.

2.2.3 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.

2.2.4 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.

2.2.5 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.

2.3.4 Check the internationalized string for disallowed names

If the internationalized string consists only of characters that conform to the host name requirements in [STD13], the conversion MUST stop with an error.

2.4 Compression algorithm

The basic method for compression is to reduce a full string that consists of characters all from a single row of the ISO 10646 repertoire, or all from a single row plus from row 0, to as few octets as possible. Any full string that has characters that come from two rows, neither of which are row 0, or three or more rows, has all the octets of the input string in the output string.

If the string comes from only one row, compression is to one octet per character in the string. If the string comes from only one row other than row 0, but also has characters only from row 0, compression is to one octet for the characters from the non-0 row and two octets for the characters from row 0. Otherwise, there is no compression and the output is a string that has two octets per input character.

The compressed string always has a one-octet header. If the string comes from only one row, the header octet is the upper octet of the characters. If the string comes from only one row other than row 0, but also has characters only from row 0, the header octet is the upper octet of the characters from the non-0 row. Otherwise, the header octet is 0xD8, which is the upper octet of a surrogate pair. Design note: It is impossible to have a legal stream of UTF-16 characters that has all the upper octets being 0xD8 because a character whose upper octet is 0xD8 must be followed by one whose upper octet is in the range 0xDC through 0xDF.

Although the two-octet mode limits the number of characters in a RACE 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.

Design note: In step 5, 0xFF was chosen as the escape character because it appears in the fewest number of scripts in ISO 10646, and therefore the "escaped escape" will be needed the least. 0x99 was chosen as the second octet for the "escaped escape" because the character U+0099 has no value, and is not even used as a control character in the C1 controls or in ISO 6429.

1) Starting at the beginning of the input, read each pair of octets in the input stream, comparing the upper octet of each. Reset the input pointer to the beginning of the input again. If all of the upper octets (called U1) are the same, go to step 4. Note that if the input is only one character, this test will always be true.

2) Read each pair of octets in the input stream, comparing the upper octet of each. Reset the input pointer to the beginning of the input again. If all of the upper octets are either 0x00 or one single other value (called U1), go to step 4.

3) Output 0xD8, followed by the entire input stream. Finish.

4) If U1 is in the range 0xD8 to 0xDC, stop with an error. Otherwise, output U1.

5) If you are at the end of the input string, finish. Otherwise, read the next octet, called U2, and the octet after that, called N1. If U2 is 0x00 and N1 is 0x99, stop with an error.

6) If U2 is equal to U1, and N1 is not equal to 0xFF, output N1, and go to step 5.

7) If U2 is equal to U1, and N1 is equal to 0xFF, output 0xFF followed by 0x99, and go to step 5.

8) Output 0xFF followed by N1. Go to step 5.

2.4.2 Decompressing a string

1) Read the first octet of the input string. Call the value of the first octet U1. If there are no more octets in the input string (that is, if the input string had only one octet total), stop with an error. If U1 is 0xD8, go to step 8.

2) If you are at the end of the input string, go to step 11. Otherwise, read the next octet in the input string, called N1. If N1 is 0xFF, go to step 5.

3) If U1 is 0x00 and N1 is 0x99, stop with an error.

4) Put U1 followed by N1 in the output buffer. Go to step 2.

5) If you are at the end of the input string, stop with an error.

6) Read the next octet of the input string, called N1. If N1 is 0x99, put U1 followed by 0xFF in the output buffer, and go to step 2.

7) Put 0x00 followed by N1 in the output buffer. Go to step 2.

8) Read the rest of the input stream into a temporary string called LCHECK. If the length of LCHECK is an odd number, stop with an error.

9) Perform the checks from steps 1 and 2 of the compression algorithm in section 2.4.1 on LCHECK. If either checks pass (that is, if either would have created a compressed string), stop with an error because the input to the decompression is in the wrong format.

10) If the length of LCHECK is odd, stop with and error. Otherwise, output LCHECK and finish.

11) If the length of the output buffer is odd, stop with and error. Otherwise, emit the output buffer and finish.

2.4.3 Compression examples

For the input string of <U+012D><U+0111><U+014B>, all characters are in the same row, 0x01. Thus, the output is 0x012D114B.

For the input string of <U+012D><U+00E0><U+014B>, the characters are all in row 0x01 or row 0x00. Thus, the output is 0x012DFFE04B.

For the input string of <U+1290><U+12FF><U+120C>, the characters are all in row 0x12. Thus, the output is 0x1290FF990C.

For the input string of <U+012D><U+00E0><U+24D3>, the characters are from two rows other than 0x00. Thus, the output is 0xD8012D00E024D3.

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) If the input bit stream is not an even multiple of five bits, pad the input stream with 0 bits until it is an even multiple of five bits. Set the read pointer to the beginning of the input bit stream.

2) Look at the five bits after the read pointer.

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.

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) Count the number of octets in the input and divide it by 8; call the remainder INPUTCHECK. If INPUTCHECK is 1 or 3 or 6, stop with an error.

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

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

4) Move the read pointer one octet forward. If the read pointer is not at the end of the input octet stream (that is, there are more octets in the input), go to step 3.

5) Count the number of bits that are in the output buffer and divide it by 8; call the remainder PADDING. If the PADDING number of bits at the

end of the output buffer are not all zero, stop with an error. Otherwise, emit the output buffer and stop.

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, RACE 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.

RACE 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

Mark Davis contributed many ideas to the initial draft of this document, as well as comments in later versions. Graham Klyne and Martin Duerst offered technical comments on the algorithms used. GIM Gyeongseog and Pongtorn Jentaweepornkul helped fix technical errors in early drafts. Rick Wesson and Mark Davis contributed many suggestions on error conditions in the processing.

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

B. Changes from Versions -02 to -03 of this Draft

1: Wording corrections to third paragraph.

2.2 and 2.3: Added need to check for all-STD13.

2.4.1: Wording corrections in the first two paragraphs. Made step 1 and **2 clearer with resetting the input pointer. Also added sentence at the** end of step 1. Also added error conditions in steps 4 and 5.

2.4.2: Added error condition in step 1. Added a new step 3 for an error check. Expanded step 8 to check for malformed input error. Added error check for odd-length output.

2.4.3: Changed all the examples to use lowercase characters on input.

2.5.1: Made the list of steps shorter by padding with 0 bits at the beginning of the steps.

2.5.2: Changed the sense of the test in step 3 and added step 4 to be checkfor malformed input. Also made the output a buffer. Also added new step 1.

C. IANA Considerations

There are no IANA considerations in this document.

D. Author Contact Information

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