RACE: Row-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."

To view the list Internet-Draft Shadow Directories, see 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 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 10646 [ISO10646] coded character set 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 18 characters, and names in most other scripts can have up to 36 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 36 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 36 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 18 characters per name part are allowed.

- If the characters in the input come from two rows, one of which is row 0, between 18 and 35 characters per name part are allowed.

1.3 Open issues

Is it OK in 2.3.2 to say "0 MAY be converted to 0 and that 1 MAY be converted to 1"?
Do we want to leave the unused characters 0, 1, 8, and 9 "reserved" in Base32 instead of making them prohibited now? This allows creative expansion in the future.

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 "ra--". (Of course, because host name parts are case-insensitive, this might also be represented as "Ra--" or "rA--" or "RA--"). The string "ra--" was chosen because it is extremely unlikely to exist in host parts before this specification was produced. As a historical note, in mid-April 2000, none of the second-level host name parts in any of the .com, .edu, .net, and .org top-level domains began with "ra--"; there are about 36,000 other strings of three characters followed by a hyphen that have this property and could be used instead.

Note that a zone administrator might still choose to use "ra--" 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 "ra--" unless those names are post-converted names. Creating host part names that begin with "ra--" 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 performed in the exact order of the subsections given here.

Note that if any checking for prohibited name parts (such as ones that are already legal DNS name parts), prohibited characters, case-folding, or canonicalization is to be done, it MUST be done before doing the conversion to an ACE name part. (Previous versions of this draft specified these steps.)

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 (that is, outside the first 0xFFFF characters) 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 "ra--" to the encoded string and finish

Prepend the characters "ra--" 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 ones that are already legal DNS name parts), prohibited characters, case-folding, or canonicalization is to be done, it MUST be done after doing the conversion from an ACE name part. (Previous versions of this draft specified these steps.)

2.3.1 Strip the "ra--"

The input string MUST begin with the characters "ra--". If it does not, the conversion MUST stop with an error. Otherwise, remove the characters "ra--" 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 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 18, 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 choose 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

Design note: No checking is done on the input to this algorithm. It is assumed that all checking for valid ISO 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) Read each character in the input stream, comparing the upper octet of each. If all of the upper octets (called U1) are the same, go to step 4.

2) Read each character in the input stream, comparing the upper octet of each. If all of the upper octets are either 0 or one single other value (called U1), go to step 5.

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

4) Output U1. Output the lower octet of each character in the input. Finish.

5) Output U1.

6) 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.

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

8) If U2 is equal to U1, and N1 is equal to 0xFF, output 0xFF followed by 0x99, and go to step 6.

9) Output 0xFF followed by N1. Go to step 6.

2.4.2 Decompressing a string
1) Read the first octet of the input string. Call the value of the first octet U1. If U1 is 0xD8, go to step 7.

2) If you are at the end of the input string, finish. Otherwise, read the next octet in the input string, called N1. If N1 is 0xFF, go to step 4.

3) Output U1 followed by N1. Go to step 2.

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

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

6) Output 0x00 followed by N1. Go to step 2.

7) Read the rest of the input stream and put it in the output stream. Finish.

2.4.3 Compression examples

For the input string of <U+012E><U+0110><U+014A>, all characters are in the same row, 0x01. Thus, the output is 0x012E104A.

For the input string of <U+012E><U+00D0><U+014A>, the characters are all in row 0x01 or row 0x00. Thus, the output is 0x012EFFD04A.

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+012E><U+00D0><U+24C3>, the characters are from two rows other than 0x00. Thus, the output is 0xD8012E00D024C3.

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

\[
\begin{array}{ccccccc}
3 & a & 2 & 7 & 0 & f & 9 & 3 \\
0011010 & 00100111 & 00001111 & 10010011
\end{array}
\]

Broken into chunks of five bits, this is:

\[
\begin{array}{cccccccc}
00111 & 01000 & 10011 & 10000 & 11111 & 00100 & 11 \\
& 00111 & 01000 & 10011 & 10000 & 11111 & 00100 & 11000
\end{array}
\]

Padding is added to make the last chunk five bits:

\[
\begin{array}{cccccccc}
00111 & 01000 & 10011 & 10000 & 11111 & 00100 & 11000 \\
& 00111 & 01000 & 10011 & 10000 & 11111 & 00100 & 11000
\end{array}
\]

The output of encoding is:

\[
\begin{array}{ccccccc}
h & i & t & q & 7 & e & y \\
00111 & 01000 & 10011 & 10000 & 11111 & 00100 & 11000
\end{array}
\]

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


[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 ]]]


A. Acknowledgements

Mark Davis contributed many ideas to the initial draft of this document. Graham Klyne and Martin Duerst offered technical comments on the algorithms used.

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

B. Changes from Previous Versions of this Draft

B.1 Changed from -03 to -04

This version of the document is radically changed to make it just a template for an ACE, not a potential full IDN protocol. I believe that a combination protocol that uses both binary on the wire and
an ACE is a better solution than an ACE-only protocol.

Title: Changed completely.

Abstract: Reworded completely.

Throughout: changed "aq8" to "ra--".

Throughout: changed "domain name" to "host name" where appropriate (which was almost everywhere).

1: Reworded the beginning to narrow the scope.

1.2: Added this section.

1.3: Added the "open issues" section.

2: Moved the first paragraph up to section 1.

2.1: Added discussion of rejection problems with improper name tagging.

2.2: Removed all pre-checking, and put this into the process that calls RACE.

2.2.1: Removed.

2.2.2: Removed.

2.2.3: Removed. Renumbered 2.2.4 through 2.2.7 to 2.2.1 through 2.2.4.

2.2.5 (old): Changed the values to 36 to reflect the correct maximum.

2.2.6 (old): Shortened the first sentence.

2.3: Removed all post-checking, and put this into the process that calls RACE.

2.3.1: Changed to make failure here an error.

2.3.2: Changed to make failure here an error.

2.3.4: Removed.

2.4: Changed the algorithm to be better optimized for strings that come from one row plus row 0. This caused a change in almost everything in 2.4, 2.4.1, and 2.4.2.

2.4.3: Added this section of examples.

2.5: Renumbered Table 2 to Table 1.

2.5.3: Added padding step to the example.

3: Removed entire section. Renumbered 4 (Security Considerations) to 3 and renumbered 5 (References) to 4.

5: Added [IDNComp]. Removed [Norm]. Removed [RFC2278] and [UnicodeData].

B.2 Changes from -02 to -03

Throughout: changed "wg4" to "aq8".

2.2: Updated the first design note to indicate that the table
will probably be moved to its own draft.

2.2.3: Changed reference for normalization from [UTR15] to [Norm].

5: Updated the reference for [IDNReq]. Removed [UTR15] and replaced it with [Norm].

B.3 Changes from -01 to -02

Throughout: Changed "ph6" to "wg4".

2.1: Updated count of unused three-letter prefixes.

2.3: Removed all the error states and clarified that any error in conversion means that the input string is the post-converted string.

2.4: Radically changed the compression scheme; the previous one was far too cumbersome.

2.5: Renumbered Table 3 to Table 2.

2.5.1: Changed the second paragraph (should have been done in the change to -01 to remove padding).

3.2: Clarified the paragraph emphasizing the need for users to be able to copy names even if they are not displayable.

5: Removed reference to [UTR6].

A: Added Martin Duerst. Removed reference to the compression algorithm because it has changed.

B.4 Changes from -00 to -01

Throughout: Changed references to the character set from Unicode to ISO 10646, even though they are equivalent. Also changed references to the rules for surrogate pairs to ISO 10646.

1.1: Clarified last paragraph.

2.2: Reworded the first design note to make excluding case stuff more likely.

2.5: Removed the "8" padding in the Base32 algorithm because it was superfluous.

2.5.1: Removed "in network byte order" from the first sentence because it was redundant.

3.3: Made the first paragraph stronger.

5: Added reference to ISO 10646. This still needs work.

A: Added Graham Klyne.

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