Internet Draft   Edmon Chung, Neteka Inc.
<draft-ietf-idn-ace16x-00.txt>   David Leung, Neteka Inc.
                                                               June 2001

ACE using Extended Hex Values (ACE16x)

STATUS OF THIS MEMO

Abstract

ACE16x is a simplified version of DUDE [DUDE-02] that requires no 5 bit or base-32 mapping. ACE16x encoding results in a string that performs as well as DUDE technically.

Instead of resorting to a quartet-to-quintet mapping mechanism, ACE16x simply uses the hex values with an extended hex (16x) scheme for compression. In essence, instead of pre-pending an extra bit, ACE16x shifts the last quartet of a compressed code point up to another character. Additionally, the 16x value is calculable instead of needing to be mapped.

Terminology

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

LDH: Letters, Digits and Hyphens: a string of characters that consists only hyphens ("-"), English letters (A-z) and digits (0-9), which might not be a result of an algorithm for transcoding multilingual characters. For example: whatever-you-want.example

ACE - ASCII Compatible Encoding: a string of characters resulting from a particular algorithm for transforming multilingual character information into an alphanumeric form acceptable by the existing DNS. For example: bq--3bhc2zmh.tld. In essence, ACE is a subset of LDH.

Hexadecimal values are shown preceeded by "0x". For example, 0x60 is decimal 96. As in the Unicode Standard [UNICODE], Unicode code points are denoted by "U+" followed by four to six hexadecimal digits, while a range of code points is denoted by two hexadecimal numbers separated by "..", with no prefixes.

Table Of Contents

1. Introduction

ACE16x is very similar to DUDE. Except that it does not require any base-32 mapping.

For example, the Unicode sequence (Sections 2-4 will further discuss the algorithm):

```
   Char:      <gumi>            <kinpachi>          <sensei>
--------------------------------------------------------------------
Unicode:       U+516B              U+5148              U+751F
    Bin: 0101 0001 0110 1011 0101 0001 0100 1000 0111 0101 0001 1111
--------------------------------------------------------------------
 ACE16x: 0101 0001 0101 1011           0010 0011 0010 0100 0101 0111
(in Bin)
 ACE16x:    5    1    5    r              2    j    2    4    5    n
(in LDH with last quartet of each code point shifted to 16x)
--------------------------------------------------------------------
   DUDE:10101100011000001011          1001000011100101010010101001111
(in Bin)
   DUDE:    x    t    s    m              u    d    u    w    x    h
(in LDH prepending 1 & 0s and mapping to base32)
```

```
In brief:
    ACE16x: 515r2j245n
      DUDE: xtsmuduwxh
```

Lengthwise, ACE16x is exactly the same as DUDE, while ACE16x does not require any 5 bit handling and mapping. This largely simplifies and speeds up the process as compared with DUDE.

2. Extended Hex Values (16x)

The extended hex (16x) values are used for the final quartet of a compressed code point. This is used to preserve the reversibility of the encoded string, without compromising length while avoiding having to do a base-32 mapping.

There are 16 characters used for hex values, the following table provides the extended hex (16x) values for each hex digit.

```
Hex=Bin=16x
0=0000=G   1=0001=H   2=0010=I   3=0011=J
4=0100=K   5=0101=L   6=0110=M   7=0111=N
8=1000=O   9=1001=P   A=1010=Q   B=1011=R
C=1100=S   D=1101=T   E=1110=U   F=1111=V
```

Note that the characters are shifted exactly 16 alphabetic positions from their original hex value. Therefore no mapping is required. The 16x value could be calculated:

```
16x value = Original hex value + 0x67 (or +0x47 for uppercase*)
     *0x67 is the code value for the lowercase letter "g".
      0x47 is the code value for the uppercase letter "G".
```

Unless the "mixed-case annotation" feature is implemented, lowercase or uppercase form is accepted. Since all 16x values are letters, for mixed-case annotations, an uppercase 16x value indicates an uppercase character and vice versa (Appendix B).

3. Encoding Procedure

Similar to DUDE, all ordering of bits and quartets is big-endian (most significant first).

```
let prev = 0x30
for each input integer n (in order) do begin
   if n == 0x2D then output hyphen-minus
   else begin
      let diff = prev XOR n
      hex dump resulting quartets,
         as few as are sufficient (but at least one), and
         shift the last quartet to its 16x value
```

```
        let prev = n
      end
    end
```

Nameprep [NAMEPREP] is not discussed in this document, but is expected that it be implemented for IDN. Hence, regardless of the code point presented, an encoder MUST not produce an incorrect output. The encoder must fail if it encounters a negative input value.

The initial value used is 0x30 so that all domains beginning with a digit will be shorter.

4. Decoding Procedure

```
let prev = 0x30
while the input string is not exhausted do begin
   if the next character is hyphen-minus
   then consume it and output 0x2D
   else begin
      consume characters and convert them to quartets until
         encountering a 16x value
      fail upon encountering a non-ACE16x character (0-v)
         or end-of-input
      shift the 16x value back to its hex form
      concatenate the resulting quartets to form diff
      let prev = prev XOR diff
      output prev
   end
end
encode the output sequence and compare it to the input string
fail if they do not match (case insensitively)
```

5. Implementation & Examples

The following examples illustrates the similarities and differences between dude:

```
(A) Unicode: U+0031
     ACE16x: h
       DUDE: xb
```

Note that with Nameprep both should be "1" since the entire label consists of LDH only. This is just to show how the initial diff (0x30) value affects the resulting string.

All of the following examples are taken from the DUDE-02 draft:

```
(B) Unicode: U+2C7EF U+2C7EF
     ACE16x: 2c7dvg
       DUDE: u6z2ra

(C) Unicode: U+1752B U+1752A
     ACE16x: 1751rh
```

DUDE: tzxwmb

```
       (D) Unicode: U+63AB1 U+63ABA
            ACE16x: 63a8hr
              DUDE: yv47bm

       (E) Unicode: U+261AF U+261BF
            ACE16x: 2619v1g
              DUDE: uyt6rta

       (F) Unicode: U+C3A31 U+C3A8C
            ACE16x: c3a0hbt
              DUDE: 6v4xb5p

       (G) Unicode: U+09F44 U+0954C
            ACE16x: 9f7ka0o
              DUDE: 39ue4si

       (H) Unicode: U+8D1A3 U+8C8A3
            ACE16x: 8d19j190g
              DUDE: 27t6dt3sa

       (I) Unicode: U+6C2B6 U+CC266
            ACE16x: 6c28ma00dg
              DUDE: y6u7g4ss7a

       (J) Unicode: U+002D U+002D U+002D U+E848F
            ACE16x: ---e84bv
              DUDE: ---82w8r

       (K) Unicode: U+BD08E U+002D U+002D U+002D
            ACE16x: bd0bu---
              DUDE: 57s8q---

       (L) Unicode: U+A9A24 U+002D U+002D U+002D U+C05B7
            ACE16x: a9a1k---69f9j
              DUDE: 434we---y393d

       (M) Unicode: U+7FFFFFFF
            ACE16x: 7fffffcv or explicit failure
              DUDE: z999993r or explicit failure

       (N) 3<nen>b<gumi><kinpachi><sensei>  (Latin, kanji)
           Unicode: U+0033 U+5E74 U+0062 U+7D44 U+91D1 U+516B U+5148
                    U+751F
            ACE16x: j5e4n5e1m7d2mec9lc0bq2j245n
              DUDE: xdx8whx8tgz7ug863f6s5kuduwxh

       (O) <amuro><namie>-with-super-monkeys  (Latin, kanji, hyphens)
           Unicode: U+5B89 U+5BA4 U+5948 U+7F8E U+6075 U+002D U+0077
                    U+0069 U+0074 U+0068 U+002D U+0073 U+0075 U+0070
                    U+0065 U+0072 U+002D U+006D U+006F U+006E U+006B
```

```
              U+0065 U+0079 U+0073
      ACE16x: 5bbp2t2es26cm1ffr-600i1u1t1s-1rml1l1n-1vihlu1sq
        DUDE: x58jupu8nuy6gt99m-yssctqtptn-tmgftfth-trcbfqtnk
```

```
   (P) maji<de>koi<suru>5<byou><mae>  (Latin, hiragana, kanji)
       Unicode: U+006D U+0061 U+006A U+0069 U+3067 U+006B U+006F
                U+0069 U+3059 U+308B U+0035 U+79D2 U+524D
        ACE16x: 5tsrj300u300skm303gdi30bu79en2b9v
          DUDE: pnmdvssqvssnegvsva7cvs5qz38hu53r

   (Q) <pafii>de<runba>  (Latin, katakana)
       Unicode: U+30D1 U+30D5 U+30A3 U+30FC U+0064 U+0065 U+30EB
                U+30F3 U+30D0
        ACE16x: 30ehk7m5v309oh308u1o2j
          DUDE: vs5bezgxrvs3ibvs2qtiud

   (R) <sono><supiido><de>  (hiragana, katakana)
       Unicode: U+305D U+306E U+30B9 U+30D4 U+30FC U+30C9 U+3067
        ACE16x: 306t3jdn6t2o3lau
          DUDE: vsvpvd7hypuivf4q
```

6. Key Improvements of ACE16x in comparison with DUDE-02

- ACE16x does NOT need character mapping. Instead it uses a shifting mechanism that is calculable:

  16x = Original hex + 0x67 (or +0x47 for uppercase)

- ACE16x maintains the one pass system and utilizes XOR instead of masking as in DUDE-01

- ACE16x does not employ a 5bit mechanism, therefore increases efficiency

- The initial value is set to 0x30 so that all domains beginning with a digit will be shorter when encoded

- ACE16x simply hex dumps most quartets improving process time both in encoding and decoding.

- The overall process time will be reduced by means of the following:
    1) Hex dump verses base-32 mapping
    2) Shifting verses base-32 mapping
    3) No need to pre-pend "1" or "0" bit(during encode)
    4) No need to strip first bit (during decode)

- ACE16x is a much more simple algorithm without compromising performance. The encoding mechanism is so simple that it could easily be expressed in an Excel spreadsheet: http://www.dnsii.org/ace16x/ace16x-encode.xls (The DUDE encode mechanism is also represented in a separate worksheet. It could be observed that ACE16x is much more simple than DUDE.)

7. Security Considerations

This document does not talk about DNS security issues, and it is believed that the proposal does not introduce additional security problems not already existent and/or anticipated by adding multilingual characters to DNS and/or using ACE.

8. References

[Nameprep]Paul Hoffman, IMC & VPNC & Marc Blanchet, ViaGenie, "Preparation of Internationalized Host Names", February 24, 2001

[DUDE-02] Mark Welter, Brian W. Spolarich & Adam M. Costello,"Differential Unicode Domain Encoding (DUDE)", June 7, 2001.

Appendix A. Acknowledgements

The ACE16x draft is largely based on DUDE-02. The authors would like to thank the authors of DUDE-02 Mark Welter, Brian W. Spolarich & Adam M. Costello for their inspiration.

Appendix B. Mixed-case annotation

This section is taken from DUDE and modified for ACE16x

In order to use ACE16X to represent case-insensitive Unicode strings, higher layers need to case-fold the Unicode strings prior to ACE16X encoding. The encoded string can, however, use mixed-case 16x as an annotation telling how to convert the folded Unicode string into a mixed-case Unicode string for display purposes.

Each Unicode code point (unless it is U+002D hyphen-minus) is represented by a sequence of hex and 16x characters, the last of which is always a 16x character, which is always a letter (as opposed to a digit). If that letter is uppercase, it is a suggestion that the Unicode character be mapped to uppercase (if possible); if the letter is lowercase, it is a suggestion that the Unicode character be mapped to lowercase (if possible).

ACE16X encoders and decoders are not required to support these annotations, and higher layers need not use them.

Example: In order to suggest that example (O) in Section 5 "Implementation & Examples" be displayed as:

```
<amuro><namie>-with-SUPER-MONKEYS
```

one could capitalize the ACE16X encoding as:

5bbp2t2es26cm1ffr-600i1u1t1s-1RML1L1N-1VCBLU1SQ

Authors:

Edmon Chung
Neteka Inc.
2462 Yonge St. Toronto,
Ontario, Canada M4P 2H5
edmon@neteka.com

David Leung
Neteka Inc.
2462 Yonge St. Toronto,
Ontario, Canada M4P 2H5
david@neteka.com