Internet Draft
draft-ietf-idn-mace-00.txt
Jun 21, 2001
Expires Dec 21, 2001

M. Ishisone
SRA
Y. Yoneya
JPNIC

MACE: Modal ASCII Compatible Encoding for IDN

Status of this Memo

Abstract

MACE is a reversible transformation method from a sequence of Unicode [UNICODE] characters to a sequence of ASCII letters, digits and hyphens (LDH characters). It is designed to be used as an encoding for internationalized domain names [IDN].

Contents

1. Introduction

MACE is intended to be used as an ACE in the IDNA architecture [IDNA], and encodes a sequence of Unicode (ISO/IEC 10646) characters in the range U+0000-U+10FFFF as a sequence of LDH characters.

MACE is designed to have following features:

Completeness: Every Unicode string has a map to an LDH character string.

Uniqueness: Every Unicode string maps to at most one LDH character string.

Reversibility: The original Unicode string can be obtained from an LDH character string to which the Unicode string maps.

Efficiency: The ratio of encoded size to original size is small. If the code points of the Unicode string are clustered, a compression algorithm enables a compact encoding. Even if they are not, the encoded size is still kept small.

Simplicity: The encoding/decoding algorithms are fairly simple to implement.

2. Terminology

LDH characters are the letters A-Z and a-z, the digits 0-9, and hyphen-minus.

As in the Unicode Standard [UNICODE], Unicode characters are denoted by "U+" followed by four to six hexadecimal digits representing its UCS-4 code point. A range of Unicode characters is denoted by the form "U+xxxx-U+yyyy".

3. Overview

MACE encodes a sequence of Unicode (ISO/IEC 10646) characters in the

range U+0000-U+10FFFF as a sequence of LDH characters.

MACE is a modal encoding. There are two major modes and one of which has four submodes. Each character is encoded in a specific mode/submode. The mode/submode is chosen according to the code point

of the character and possibly its neiboring characters. The modal encoding enables compact representation of each character, and the modes are chosen so that mode change occurs rather infrequently as long as the source string is written in a single language.

LDH characters are represented literally, for the compactness of the encoded result. Other Unicode characters are represented as base32 format strings. Each of Unicode characters in Basic Multilingual Plane (BMP, U+0000-U+FFFF) except LDH characters is encoded as a 3-octet base32 format sting, while each non-BMP (U+10000-U+10FFFF) character is encoded as a 4-octet base32 format string.

To achieve fairly good compression for non-LDH charactes, there is also a submode for differential encoding. Using this submode, characters are encoded as bitwise-xor value between the code points of the previous character and the current character. In this submode a character is encoded as a 1 or 2 octet base32 format string.

So if the code points of the input string are clusterd in a small region, an effective compression algorithm enables 1 or 2 octets/character encoding (plus some overhead for mode changes). Even if the code points are widely scattered and difficult to compress (such as CJK Han characters), 3 octets/character (for BMP) or 4 octets/character (for Non-BMP) encoding (plus some overhead for mode changes) can be achieved.

4. Base32 Format

MACE uses base32 format string to encode non-negative intergers. The base32 format used for MACE is:

"0" = 0 = 0x00 = 00000	"g" = 16 = 0x10 = 10000
"1" = 1 = 0x01 = 00001	"h" = 17 = 0x11 = 10001
"2" = 2 = 0x02 = 00010	"i" = 18 = 0x12 = 10010
"3" = 3 = 0x03 = 00011	"j" = 19 = 0x13 = 10011
"4" = 4 = 0x04 = 00100	"k" = 20 = 0x14 = 10100
"5" = 5 = 0x05 = 00101	"l" = 21 = 0x15 = 10101
"6" = 6 = 0x06 = 00110	"m" = 22 = 0x16 = 10110
"7" = 7 = 0x07 = 00111	"n" = 23 = 0x17 = 10111

```
"8" =  8 = 0x08 = 01000      "o" = 24 = 0x18 = 11000
"9" =  9 = 0x09 = 01001      "p" = 25 = 0x19 = 11001
"a" = 10 = 0x0A = 01010      "q" = 26 = 0x1A = 11010
"b" = 11 = 0x0B = 01011      "r" = 27 = 0x1B = 11011
"c" = 12 = 0x0C = 01100      "s" = 28 = 0x1C = 11100
"d" = 13 = 0x0D = 01101      "t" = 29 = 0x1D = 11101
"e" = 14 = 0x0E = 01110      "u" = 30 = 0x1E = 11110
"f" = 15 = 0x0F = 01111      "v" = 31 = 0x1F = 11111
```

The encoding is big-endian (most-significant bits first). The following shows some examples.

decimal	hexadecimal	binary	base32 string
40	0x28	00001 01000	"18"
9876	0x2694	01001 10100 10100	"9kk"

5. Notations

In the following description, following five functions are used.

base32_encode(N, LEN)
: denotes a base32 format string of LEN octets representing number N. If LEN is larger than what needs to represent N, "0" is prepended.

base32_decode(S)
: denotes a number which corresponds to a base32 format string S.

codepoint(C)
: denotes a UCS-4 code point value for character C.

character(N)
: denotes a Unicode character whose UCS-4 code point is N.

xor(N, M)
: denotes a bit-wise XOR value of integer N and M.

6. Encoding Description

MACE can encode Unicode/ISO10646 characters in the range U+0000-U+10FFFF. If the input string contains other characters, or it represents a non-internationalized host name parts (conforms to

[STD13]), it MUST NOT be converted.

MACE has several encoding modes/submodes. There are two major modes, `Literal' and `Non-Literal'. Non-Literal mode has four submodes, while Literal mode has none. Each character is encoded in a specific mode/submode. The encoding process of a character is:

1. Determine the mode/submode to encode the character.
2. If and only if it is necessary to change the current mode, output ASCII hyphen-minus to change the mode.
3. If and only if it is necessary to change the current submode, output the submode introducer octet (described below) to change the submode.
4. Encode the character in the mode/submode.

ASCII letter and digit characters are encoded in Literal mode, while non-LDH characters are encoded in Non-Literal mode. ASCII hyphen

character (U+002D) can be encoded in either modes, and is always encoded as a sequence of two hyphen-minus ("--"). Switching between Literal mode and Non-Literal mode is indicated by an ASCII hyphen not followed by another hyphen. The initial mode is Non-Literal.

In Literal mode, characters are encoded as they are. For example ASCII character "a" is encoded as "a". In Non-Literal mode, characters are encoded as a base32 format string.

Non-Literal mode further comprises four submodes, `BMP-A', `BMP-B', `Non-BMP' and `Compress'. Every non-LDH character is encoded one of these submodes. Shifting to each submode is indicated by a certain octet (called introducer octet) shown below. These introducer octets can be distinguished from the base32 string since they never appear in the base32 string used by MACE.

submode	introducer octet
BMP-A	"w"
BMP-B	"x"
Non-BMP	"y"
Compress	"z"

Switching between Literal mode and Non-Literal mode doesn't affect current submode, that is, on returning from the Literal mode, previous submode is restored. This lowers the necessity of submode changes. The initial submode is BMP-A.

BMP-A and BMP-B submodes are used for encoding characters in Unicode Basic Multilingual Plane (U+0000-U+FFFF), except LDH characters. In these submodes, a character is encoded as base32 format string of 3 octets. BMP-A is used for characters in the range U+0000-U+1FFF and U+A000-U+FFFF, covering most of Western/Middle-Eastern scripts and Hangul. BMP-B is used for characters in the range U+2000-U+9FFF, covering CJK unification area. Those characters are first mapped to integers of the range 0x0000-0x7fff (15bit integer), then converted to base32 format string using the following scheme:

submode	character range	encoding
BMP-A	U+0000-U+1FFF	base32_encode(codepoint(C), 3)
	U+A000-U+FFFF	base32_encode(codepoint(C) - 0x8000, 3)
BMP-B	U+2000-U+9FFF	base32_encode(codepoint(C) - 0x2000, 3)

Here are some examples:

character	submode	integer	base32 string
U+00B0	BMP-A	0xb0	"05g"
U+5678	BMP-B	0x3678	"djo"
U+BCDE	BMP-A	0x3CDE	"f6u"

Non-BMP submode is used for encoding Unicode characters outside Basic Multilingual Plane (U+10000-U+10FFFF). In this mode a character is encoded as base 32 format string of 4 octets. Characters U+10000-U+10FFFF are first mapped to intergers of the range 0x00000-0xFFFFF (20bit integer), then converted to bae32 format string using the following scheme:

submode	character range	encoding
Non-BMP	U+10000-U+10FFFF	base32_encode(codepoint(C) - 0x10000, 4)

Compress submode is used for the efficient encoding of non-LDH characters. This mode can be used for any non-LDH characters if certain condition is met. In this mode, a character is encoded as a bit-wise XOR value between the code point of the character (called C)

and the last non-LDH character before C (called PREV). The XOR value (xor(codepoint(PREV), codepoint(C))) must be less than 0x200, or the Compress submode cannot be used. If the XOR value is less than 16, it is encoded as a base32 format string of 1 octet. Otherwise 0x200 is added to the XOR value, then it is encoded as a base32 format string of 2 octets. When decoding, this encoding enables to determine the encoded length by looking at the first octet.

submode	character range	encoding	condition
Compress	U+0000-U+10FFFF	base32_encode(X, 1)	if X < 16
		base32_encode(X + 0x200, 2)	if X >= 16

[where X is xor(codepoint(PREV), codepoint(C))]

There are two possible submodes for encoding a non-LDH character C, one of which is Compress, and the other is one of the other three (BMP-A, BMP-B, Non-BMP). The submode is determined using the following algorithm. This algorithm is designed so that it chooses the submode which produces shorter encoding result.

1. Let PREV be the last non-LDH character before C, and let NXT be the first non-LDH character after C. In case C is the first non-LDH character of the input string, let PREV be U+0000.
2. If xor(codepoint(PREV), codepoint(C)) > 0x1FF, go to 4.
3. If at least one of the following conditions holds, choose `Compress'. Otherwise go to 4.
   a) the current submode is `Compress'
   b) C is non-BMP character (U+10000-U+10FFFF)

   c) xor(codepoint(PREV), codepoint(C)) is less than 16
   d) NXT exists and xor(codepoint(C), codepoint(NXT)) <= 0x1ff
4. If C is in the range U+0000-U+1FFF or U+A000-U+FFFF, choose `BMP-A'.
5. If C is in the range U+2000-U+9FFF, choose `BMP-B'.
6. Otherwise choose `Non-BMP'.

Initial state is set as follows.

```
mode    : Non-Literal
submode : BMP-A
PREV    : U+0000
```

7. Encoding Procedure

```
procedure encode(INPUT)
    MODE = `Non-Literal'
```

```
SUBMODE = `BMP-A'
PREV = U+0000

while (is_not_empty(INPUT))
    C = read_one_character(INPUT)
    if (<C is not in the range U+0000-U+10FFFF>)
        <encode error>
    else if (<C is hyphen (U+002D)>)
        output("--")
    else if (<C is ASCII letter or digit>)
        if (MODE != `Literal')
            output("-")
            MODE = `Literal'
        endif
        output(C)
    else
        if (MODE != `Non-Literal')
            output("-")
            MODE = `Non-Literal'
        endif

        if (compressible(SUBMODE, C, PREV, INPUT) == TRUE)
            NEW_SUBMODE = `Compress'
            V = xor(codepoint(PREV), codepoint(C))
            if (V >= 16)
                V = V + 0x200
                LEN = 2
            else
                LEN = 1
            endif
        else
            V = codepoint(C)
            if (0x0000 <= V <= 0x1FFF)
                NEW_SUBMODE = `BMP-A'
```



```
                LEN = 3
            else if (0xA000 <= V <= 0xFFFF)
                NEW_SUBMODE = `BMP-A'
                V = V - 0x8000
                LEN = 3
            else if (0x2000 <= V <= 0x9FFF)
                NEW_SUBMODE = `BMP-B'
                V = V - 0x2000
                LEN = 3
            else
```

```
                    NEW_SUBMODE == `Non-BMP'
                    V = V - 0x10000
                    LEN = 4
                endif
            endif
            if (NEW_SUBMODE != SUBMODE)
                output(<submode introducer for NEW_SUBMODE>)
                SUBMODE = NEW_SUBMODE
            endif
            output(base32_encode(V, LEN))
            PREV = C
        endif
    end
end

function compressible(SUBMODE, C, PREV, INPUT)
    if (xor(codepoint(C), codepoint(PREV)) > 0x1FF)
        return (FALSE)
    endif

    # The differenct between C and PREV is confined to lower 9 bits.
    if (SUBMODE == `Compress')
        return (TRUE)
    else if (codepoint(C) >= 0x10000)
        return (TRUE)
    else if (xor(codepoint(C), codepoint(PREV)) < 16)
        return (TRUE)
    else
        <peek the next non-LDH character in INPUT>
        if (<there is such a character (called NXT)> and
            xor(codepoint(NXT), codepoint(C)) <= 0x1FF)
            return (TRUE)
        endif
    endif
    return (FALSE)
end
```

8. Decoding Description

Like encoding, MACE decoding process keeps track of the current

mode/submode to decode each character. The initial state for decoding is the same as that of encoding.

```
 mode    : Non-Literal
```

```
submode : BMP-A
PREV    : U+0000
```

Because ASCII domain names are case-insensitive, decoding process MUST treat uppercase leters and lowercase letters equally.

The consecutive two ASCII hyphen-minus characters are always decoded as a single ASCII hyphen-minus, regardless of the current mode/submode. ASCII hyphen-minus not followed by another hyphen-minus indicates mode switching between Literal mode and Non-Literal mode.

In Literal mode, all ASCII letter and digit characters are decoded as they are.

In Non-Literal mode, every character is either a submode introducer or a part of base32 format string. If a character is a submode introducer, the current submode is changed to the corresponding submode. If it isn't, it is a part of base32 format string.

To decode base32 format string in a certain submode, first determine the length of the string which is decoded to a single Unicode character. For submodes other than Compress, the number of octets which encodes a character is fixed (3 for BMP-A and BMP-B, 4 for Non-BMP). For Compress submode, the number of octets is variable (1 or 2), and can be determined by looking at the first octet. If the first octet represents a number less than 16 in base32 (either 0-9, a-f or A-F) the number of octets is one, otherwise two. The following list shows the length of the string S and how to get the decoded character in each submode.

submode	length	decoded character	condition
BMP-A	3	character(N)	if N < 0x2000
		character(N + 0x8000)	if N >= 0x2000
BMP-B	3	character(N + 0x2000)	
Non-BMP	4	character(N + 0x10000)	
Compress	1	character(xor(P, N))	
	2	character(xor(P, N - 0x200))	

[where N is base32_decode(S), P is codepoint(PREV)]

MACE decoding process can accept invalidly-encoded strings as well. In order to guarantee the unique mapping, following two types of check must be performed.

1) The decoded string must be checked if it is a [STD13] conforming name. If it is, decoding process MUST fail.

2) The decoded string must be re-encoded and compared to the input string. If they are not equal (allowing case-difference), decoding process MUST fail.

9. Decoding Procedure

```
procedure decode(input)
    MODE = `Non-Literal'
    SUBMODE = `BMP-A'
    PREV = U+0000

    while (is_not_empty(INPUT))
        C = read_one_character(INPUT)
        if (<C is hyphen>)
            NXT = read_one_character(INPUT)
            if (<NXT is hyphen>)
                output("-")
            else
                <push back NXT to INPUT>
                if (MODE == `Literal')
                    MODE = `Non-Literal'
                else
                    MODE = `Literal'
                endif
            endif
        else if (MODE == `Literal')
            output(C)
        else if (<C matches one of the submode introducer octets>)
            SUBMODE = <corresponding submode>
        else
            <push back C to INPUT>
            if (SUBMODE == `BMP-A')
                S = read_string_of_length(INPUT, 3)
                V = base32_decode(S)
                if (V >= 0x2000)
                    V = V + 0x8000
                endif
            else if (SUBMODE == `BMP-B')
                S = read_string_of_length(INPUT, 3)
                V = base32_decode(S) + 0x2000
            else if (SUBMODE == `Non-BMP')
                S = read_string_of_length(INPUT, 4)
                V = base32_decode(S) + 0x10000
            else if (SUBMODE == `Compress')
                if (<C is either 0-9, a-f or A-F>)
                    S = read_string_of_length(INPUT, 1)
                    V = base32_decode(S)
```

```
else
    S = read_string_of_length(INPUT, 2)
```

```
                    V = base32_decode(S) - 0x200
                endif
                V = PREV xor V
            endif
            output(character(V))
            PREV = character(V)
        endif
    end
end
```

The above decoding procedure accepts invalidly-encoded strings as well. In order to guarantee the unique mapping, following two additional checks MUST be performed after decoding:

1) that the decoding string is NOT a [STD13] conforming name.
2) that the string which is the result of re-encoding of the decoded string matches the original string.

10. ACE Identifier

In order to use MACE as an ACE, there must be a certain prefix or suffix string which is unlikely to be used in normal domain names and thus identifies MACE-encoded domain name parts. Since MACE-encoded names can begin with hyphen-minus and names beginning with hyphen-minus do not conform [STD13], a prefix string should be used. So if MACE is used for encoding domain name parts, the encoded names should be prefixed by the prefix string.

This document does not specify the prefix string for MACE. The actual selection should be left to certain authority such as IANA [ACEID].

For testing purpose, there is a registry of test prefix strings for ACEs on IETF IDN working group web site [IDN].

11. Examples

The following examples are meaningless strings, but they are designed to exercise various aspects of the algorithm in order to verify the correctness of the implementation.

(a) U+0200 U+4000 U+002D U+B001 U+40001 U+0061
    MACE: g0x800--wc01y6001-a

(b) U+0061 U+002D U+0300 U+0062 U+0400 U+3000 U+002D U+5000
    MACE: -a---0o0-b-100x400--c00

(c) U+1FFF U+2000 U+9FFF U+A000 U+FFFF U+10000 U+10FFFF
MACE: 7vvx000vvvw800vvvy0000vvvv

(d) U+0200 U+002F U+0030 U+0039 U+003A U+0200 U+0040 U+0041 \
U+005A U+005B U+0200 U+0060 U+0061 U+007A U+007B
MACE: 0g001f-09-01q0g0020-AZ-02r0g0030-az-03r

(e) U+0061 U+0062 U+0063 U+002D U+1000 U+1200 U+002D \
U+2000 U+2010 U+2200 U+002D U+3000 U+3010
MACE: -abc---4004g0--x00000g0g0--40040g

(f) U+0100 U+0102 U+0200 U+002D U+0201 U+002D U+03FE U+0061 U+0234
MACE: zo02w0g0--z1--vv-a-ua

(g) U+3000 U+002D U+3010 U+0061 U+3100 U+310F U+31FF
MACE: x400--zgg-a-ogfng

(h) U+20000 U+002D U+20100 U+0061 U+20010 U+20012 U+200FF
MACE: y2000--zo0-a-og2nd

12. Security Considerations

Users expect each domain name in DNS to be controlled by a single authority. If a Unicode string intended for use as a domain label could map to multiple ACE labels, then an internationalized domain name could map to multiple ACE domain names, each controlled by a different authority, some of which could be spoofs that hijack service requests intended for another. Therefore MACE is designed so that each Unicode string has a unique encoding.

13. References

[UNICODE] The Unicode Consortium, "The Unicode Standard", http://www.unicode.org/unicode/standard/standard.html

[IDN] Internationalized Domain Names (IETF Working Group), http://www.i-d-n.net/, idn@ops.ietf.org

[IDNA] Patrik Falstrom, Paul Hoffman, "Internationalizing Host Names In Applications (IDNA)", draft-ietf-idn-idna-01

[STD13] Paul Mockapetris, "DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION", Nov 1987, STD 13 (RFC 1035)

[RFC952] K. Harrenstien, M. Stahl, E. Feinler, "DOD Internet Host Table Specification", Oct 1985, RFC 952

[NAMEPREP] Paul Hoffman, Marc Blanchet, "Preparation of Internationalized Host Names", Feb 2001,

[ACEID] Naomasa Maruyama, Yoshiro Yoneya, "Proposal for a determining process of ACE identifier", Jun 2001, draft-ietf-idn-aceid-02

[BRACE] Adam M. Costello, "BRACE: Bi-mode Row-based ASCII-Compatible Encoding for IDN", Sep 2000, draft-ietf-idn-brace-00

[DUDE] Mark Welter, Brian W. Spolarich, Adam M. Costello, "Differential Unicode Domain Encoding (DUDE)", Jun 2001, draft-ietf-idn-dude-02

14. Acknowlegdements

Some of the ideas in MACE are taken from other ACE proposals.

The idea of Literal/Non-Literal mode is taken from BRACE draft [BRACE] by Adam M. Costello.

The idea of differencial encoding used by Compress submode is taken from DUDE [DUDE], by Mark Welter, Brian W. Spolarich and Adam M. Costello.

The structure of this document and text of some sections are borrowed from AMC-ACE- series draft (draft-ietf-idn-amc-ace-*) by Adam M. Costello.

15. Authors' Address

Makoto Ishisone
Software Research Associates, Inc.
4-16-10, Chigasaki-Minami, Tsuzuki-ku, Yokohama,
Kanagawa 224-0037 Japan
<ishisone@sra.co.jp>

Yoshiro Yoneya
Japan Network Information Center (JPNIC)
Fuundo Bldg 1F, 1-2 Kanda-ogawamachi,
Chiyoda-ku Tokyo 101-0052, Japan
<yone@nic.ad.jp>