Internet Engineering Task Force (IETF)
INTERNET-DRAFT
draft-ietf-idn-dude-00.txt
November 16, 2000

Mark Welter
Brian W. Spolarich
WALID, Inc.

Expires May 16, 2001

DUDE: Differential Unicode Domain Encoding

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.

The distribution of this document is unlimited.

Abstract

This document describes a tranformation method for representing Unicode character codepoints in host name parts in a fashion that is completely compatible with the current Domain Name System. It provides for very efficient representation of typical Unicode sequences as host name parts, while preserving simplicity. It is proposed as a potential candidate for an ASCII-Compatible Encoding (ACE) for supporting the deployment of an internationalized Domain Name System.

Table of Contents

1. Introduction

DUDE describes an encoding scheme of the ISO/IEC 10646 [ISO10646] character set (whose character code assignments are synchronized with Unicode [UNICODE3]), and the procedures for using this scheme to transform host name parts containing Unicode character sequences into sequences that are compatible with the current DNS protocol [STD13]. As such, it satisfies the definition of a 'charset' as defined in [IDNREQ].

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".

DUDE 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". This specification defines both a forward and reverse transformation algorithm.

2. Hostname Part Transformation

According to [STD13], hostname parts must start and end with a letter or digit, and contain only letters, digits, and the hyphen character ("-"). This, of course, excludes most characters used by non-English speakers, 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 Post-Converted Name Prefix

This document defines the string 'dq--' as a prefix to identify DUDE-encoded sequences. For the purposes of comparison in the IDN Working Group activities, the 'dq--' prefix should be used solely to identify DUDE sequences. However, should this document proceed beyond draft status the prefix should be changed to whatever prefix, if any, is the final consensus of the IDN working group.

Note that the prepending of a fixed identifier sequence is only one mechanism for differentiating ASCII character encoded international domain names from 'ordinary' domain names. One method, as proposed in [IDNRACE], is to include a character prefix or suffix that does not appear in any name in any zone file. A second method is to insert a domain component which pushes off any international names one or more levels deeper into the DNS hierarchy. There are trade-offs between these two methods which are independent of the Unicode to ASCII transcoding method finally chosen. We do not address the international vs. 'ordinary' name differention issue in this paper.

2.2 Radix Selection

There are many proposed methods for representing Unicode characters within the allowed target character set, which can be split into groups on the basis of the underlying radix. We have chosen a method with radix 16 because both UTF-16 and ASCII are represented by even multiples of four bits. This allows a Unicode character to be encoded as a whole number of ASCII characters, and permits easier manipulation of the resulting encoded data by humans.

2.3 Hostname Prepartion

The hostname part is assumed to have at least one character disallowed by [STD13], and that is has been processed for logically equivalent character mapping, filtering of disallowed characters (if any), and compatibility composition/decomposition before presentation to the DUDE conversion algorithm.

While it is possible to invent a transcoding mechanism that relies

on certain Unicode characters being deemed illegal within domain names and hence available to the transcoding mechanism for improving encoding efficiency, we feel that such a proposal would complicate matters excessively. We also believe that Unicode name preprocessing for both name resolution and name registration should be considered as separate, independent issues, which we will address in a separate document.

2.4 Definitions

For clarity:

'integer' is an unsigned binary quantity;
'byte' is an 8-bit integer quantity;
'nibble' is a 4-bit integer quantity.

2.5 DUDE Encoding

The idea behind this scheme is to provide compression by encoding the contiguous least significant nibbles of a character that differ from the preceding character. Using a variant of the variable length hex encoding desribed in [IDNDUERST] and elsewhere, by encoding leading zero nibbles this technique allows recovery of the differential length. The encoding is, with some practice, easy to perform manually.

There are two extensions to this basic idea: one enables encoding the preferred case for each charcter (for reverse DNS resolution) and another improves the worse case behaviour related to surrogates. The basic algorithms will be formally described first and then the extended algorithms will be described.

2.5.1 Extended Variable Length Hex Encoding

The variable length hex encoding algorithm was introduced by Duerst in [IDNDUERST]. It encodes an integer value in a slight modification of traditional hexadecimal notation, the difference being that the most significant digit is represented with an alternate set of "digits" - -- 'g through 'v' are used to represent 0 through 15. The result is a variable length encoding which can efficiently represent integers of arbitrary length.

This specification extends the variable length hex encoding algorithm to support the compression scheme defined below by potentially not supressing leading zero nibbles.

The extended variable length nibble encoding of an integer, C, to length N, is defined as follows:

1. Start with I, the Nth least significant nibble from the least significant nibble of C;

2. Emit the Ith character of the sequence [ghijklmnopqrstuv];

3. Continue from the most to least significant, encoding each remaining nibble J by emitting the Jth character of the sequence [0123456789abcdef].

2.5.2 DUDE Compression Algorithm

1. Let PREV = 0;

2. If there are no more characters in the input, terminate successfully;

4. Let C be the next character in the input;

5. If C != '-' , then go to step 5;

6. Consume the input character, emit '-', and go to step 2;

7. Let D be the result of PREV exclusive ORed with C;

8. Find the least positive value N such that
   D bitwise ANDed with M is zero
   where M = the bitwise complement of (16**N) - 1;

9. Let V be C ANDed with the bitwise complement of M;

10. Variable length hex encode V to length N and emit the result;

11. Let PREV = C and go to step 2.

2.5.3 Forward Transformation Algorithm

The DUDE transformation algorithm accepts a string in UTF-16 [ISO10646] format as input. The encoding algorithm is as follows:

1. Break the hostname string into dot-separated hostname parts. For each hostname part which contains one or more characters disallowed by [STD13], perform steps 2 and 3 below;

2. Compress the hostname part using the method described in section 2.5.2 above, and encode using the encoding described in section 2.5.1;

3. Prepend the post-converted name prefix 'dq--' (see section 2.1 above) to the resulting string.

2.6 DUDE Decoding

2.6.1 Extended Variable Length Hex Decoding

Decoding extended variable length hex encoded strings is identical to the standard variable length hex encoding, and is defined as

follows:

```
1.  Let CL be the lower case of the first input character,

    If CL is not in set [ghijklmnopqrstuv],
      return error,
    else
      consume the input character;

2.  Let R = CL - 'g',
    Let N = 1;

3.  If no more input characters exist, go to step 9.

4.  Let CL be the lower case of the next input character;

5.  If CL is not in the set [0123456789abcdef], go to Step 9;

6.  Consume the next input character,
    Let N = N + 1;
    Let R = R * 16;

7.  If N is in set [0123456789],
      then let R = R + (N - '0')
      else let R = R + (N - 'a') + 10;

8.  Go to step 3;

9.  Let MASK be the bitwise complement of (16**N) - 1;

10.  Return decoded result R as well as MASK.
```

2.6.2 DUDE Decompression Algorithm

```
1.  Let PREV = 0;

2.  If there are no more input characters then terminate successfully;

3.  Let C be the next input character;

4.  If C == '-', append '-' to the result string, consume the character,
      and go to step 2,

5.  Let VPART, MASK be the next variable length hex decoded
      value and mask;

6.  If VPART > 0xFFFF then return error status,

7.  Let CU = ( PREV bitwise-AND MASK) + VPART,
    Let PREV = CU;

8.  Append the UTF-16 character CU to the result string;
```

9. Go to step 2.

2.6.3 Reverse Transformation Algorithm

1. Break the string into dot-separated components and apply Steps 2 through 4 to each component;

2. Remove the post converted name prefix 'dq--' (see Section 2.1);

3. Decompress the component using the decompression algorithm described above;

4. Concatenate the decoded segments with dot separators and return.

3. Examples

The examples below illustrate the encoding algorithm and provide comparisons to alternate encoding schemes. UTF-5 sequences are prefixed with '----', as no ACE prefix was defined for that encoding.

3.1 'www.walid.com' (in Arabic):

UTF-16: U+0645 U+0648 U+0642 U+0639 . U+0648 U+0644 U+064A U+062F .
U+0634 U+0631 U+0643 U+0629

DUDE: dq--m45oij9.dq--m48kqif.dq--m34hk3i9

UTF-6: wq--ymk5k8k2j9.wq--ymk8k4kaif.wq--ymj4j1k3i9

UTF-5: ----m45m48m42m39.----m48m44m4am2f.----m34m31m43m29

RACE: bq--azcuqqrz.bq--azeeisrp.bq--ay2dcqzj

LACE: bq--aqdekscche.bq--aqdeqrckf5.bq--aqddimkdfe

(more examples to come)

4. DUDE Extensions

The first extension to the DUDE concept recognizes that the first character emitted by the variable length hex encoding algorithm is always alphabetic. We encode the case (if any) of the original Unicode character in the case of the initial "hex" character. Because the DNS performs case-insensitive comparisons, mixed case international domain names behave in exactly the same way as traditional domain names. In particular, this enables reverse lookups to return names in the preferred case.

The second extension regards the treatment of Unicode surrogate characters. If surrogates are not expanded, two 16-bit surrogates are needed to represent a single codepoint in the range of 0x10000

through 0x10FFFF. This cuts the worse case limits in half for most proposals. We will assume that our input and output Unicode are in UTF-32 format -- that is, any surrogates are expanded to their UCS-4 equivalents. If the input codes all fall under 0x10000, then the extended method will emit the same length string as the basic method. One final modification takes note of the fact that the only only codepoints forcing the use of six hex digits is for those with a "10" as the fifth and sixth digits. We will encode the fifth digit using a seventeenth digit as a special case to avoid this extra expansion.

4.1 Extended DUDE Encoding

4.1.1 Modified Extended Variable Length Hex Encoding

The modified extended variable length hex encoding of an integer C to length N with case U is performed as follows:

1. If C > 0x10FFFF return error status;

2. If N < 6 go to step 5; (this is true for characters from the first 16 Planes)

3. If U is 'Uppercase' then emit 'W'
   else emit 'w'; (special case for the 17th Plane)

4. go to step 7;

5. Let I be the Nth nibble from the right of C;

6. If U is 'Uppercase'
   then emit the Ith character of sequence [GHIJKLMNOPQRSTUV],
   else emit the Ith character of sequence [ghijklmnopqrstuv];

7. Let N = N - 1;

8. Continue from N to 1, encoding each remaining nibble, J, by emitting the Jth character of sequence [0123456789abcdef].

4.1.2 Extended Compression Algorithm

1. Let PREV = 0;

2. If there are no more characters in the input, terminate successfully;

4. Let U be the case of the next character in the input;
   Let C be the lowercase value of the next input character;

5. If C != '-' , then go to step 7;

6. Consume the input character, emit '-', and go to step 2;

7. Let D be the result of PREV exclusive ORed with C;

8. Find the least positive value N such that
   D bitwise ANDed with M is zero
   where M = the bitwise complement of (16**N) - 1;

9. Let V = C ANDed with the bitwise complement of M;

10. Emit the modified variable length hex encoding of V to length
    N with case U;

11. Let PREV = C and go to step 2.

4.1.3 Extended Forward Transformation Algorithm

The overall extended encoding algorithm is as follows:

1. Break the hostname string into dot-separated hostname parts.
   For each hostname part, perform steps 2 and 3 below;

2. Compress the component using the method described in section 4.1.2 above, and encode using the encoding described in section 4.1.1;

3. Prepend the post-converted name prefix 'dq--' (see section 2.1 above) to the resulting string.

4.2 Extended DUDE Decoding

4.2.1 Modified Extended Variable Length Hex Decoding

1. Let U be the case of the next input character,
   Let C0 be the lower case of the next input character;

2. If C0 is not in set [ghijklmnopqrstuw] then return error status,
   else, consume the input character;

3. Let R = C0 - 'g'
   Let N = 1;

4. If no more input characters exist then go to step 8;

5. Let CL be the lower case of the next input character,
   If CL is not in set [0123456789abcdef] then go to step 8;

6. Consume the next input character,
   Let N = N + 1,
   Let R = R * 16,
   If CL is in set [0-9]
   then let R = R + (CL - '0')
   else let R = R + (CL - 'a') + 10;

7. Go to step 4;

8. If R < 0x100000 then go to step 10;

9. Let N = N + 1,
   If (N > 6) or (C0 != 'w')
     then return error status;

10. Let MASK be the bitwise complement of (16**N) - 1. Return
    result R, MASK, and U.

4.2.2 Extended Decompression Algorithm

1. Let PREV = 0;

2. If there are no more input characters then terminate successfully;

3. Let C be the next input character;

4. If C == '-', append '-' to the result
   string, consume the character, and go to step 2;

5. Let VPART, MASK, and U be the result of the modified extended
   variable length decoded value;

6. Let CU = (PREV 'bitwise AND' MASK) + VPART,
   Let PREV = CU;

7. If U == 'Uppercase' then let CU = the corresponding upper case value
   of CU;

8. Append CU to the result string and go to step 2.

4.2.3 Extended Reverse Transformation Algorithm

1. Break the string into dot-separated components and apply Steps
   2 through 4 to each component;

2. Remove the post converted name prefix 'dq--' (see Section 2.1);

3. Decompress the component using the extended decompression
   algorithm described in section 4.2.2 above;

4. Concatenate the decoded segments with dot separators and return.

Note that DUDE decoding will return error for input strings which do
not comply with RFC1035.

5. Security Considerations

Much of the security of the Internet relies on the DNS and any
change to the characteristics of the DNS may change the security of

much of the Internet. Therefore DUDE makes no changes to the DNS itself.

DUDE is designed so that distinct Unicode sequences map to distinct domain name sequences (modulo the Unicode and DNS equivalence rules). Therefore use of DUDE with DNS will not negatively affect security.

6. References

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

[IDNRACE] Paul Hoffman, "RACE: Row-Based ASCII Compatible Encoding for IDN", draft-ietf-idn-race;

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

[IDNNAMEPREP] Paul Hoffman and Marc Blanchet, "Preparation of Internationalized Host Names", draft-ietf-idn-nameprep;

[IDNDUERST] M. Duerst, "Internationalization of Domain Names", draft-duerst-dns-i18n;

[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;

[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

The structure (and some of the structural text) of this document is intentionally borrowed from the LACE IDN draft (draft-ietf-idn-lace-00) by Mark Davis and Paul Hoffman.

B. IANA Considerations

There are no IANA considerations in this document.

C. Author Contact Information

Mark Welter
Brian W. Spolarich
WALID, Inc.
State Technology Park
2245 S. State St.
Ann Arbor, MI 48104
+1-734-822-2020

mwelter@walid.com
briansp@walid.com
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.0.1 (GNU/Linux)
Comment: For info see http://www.gnupg.org

iD8DBQE6FZ/D/DkPcNgtD/0RAoswAKCUGBTSFJv96+Z+YnA8m47qrnheAgCeLQ6C
1+knyHluauC+66esCtPVoKU=
=hbT+
-----END PGP SIGNATURE-----