Network Working Group
Request for Comments: 4042
Category: Informational

M. Crispin
Panda Programming

1 April 2005

UTF-9 and UTF-18
Efficient Transformation Formats of Unicode

Status of This Memo

Copyright Notice

Abstract

ISO-10646 defines a large character set called the Universal Character Set (UCS), which encompasses most of the world's writing systems. The same set of codepoints is defined by Unicode, which further defines additional character properties and other implementation details. By policy of the relevant standardization committees, changes to Unicode and amendments and additions to ISO/IEC 646 track each other, so that the character repertoires and code point assignments remain in synchronization.

The current representation formats for Unicode (UTF-7, UTF-8, UTF-16) are not storage and computation efficient on platforms that utilize the 9 bit nonet as a natural storage unit instead of the 8 bit octet.

This document describes a transformation format of Unicode that takes advantage of the nonet so that the format will be storage and computation efficient.

1. Introduction

A number of Internet sites utilize platforms that are not based upon the traditional 8-bit byte or octet. One such platform is the PDP-10, which is based upon a 36-bit word. On these platforms, it is wasteful to represent data in octets, since 4 bits are left unused in each word. The 9-bit nonet is a much more sensible representation.

Although these platforms support IETF standards, many of these platforms still utilize a text representation based upon the septet,

which is only suitable for [US-ASCII] (although it has been used for various ISO 10646 national variants).

To maximize international and multi-lingual interoperability, the IAB has recommended ([IAB-CHARACTER]) that [ISO-10646] be the default coded character set.

Although other transformation formats of [UNICODE] exist, and conceivably can be used on nonet-oriented machines (most notably [UTF-8]), they suffer significant disadvantages:

[UTF-8]
: requires one to three octets to represent codepoints in the Basic Multilingual Plane (BMP), four octets to represent [UNICODE] codepoints outside the BMP, and six octets to represent non-[UNICODE] codepoints. When stored in nonets, this results in as many as four wasted bits per [UNICODE] character.

[UTF-16]
: requires a hexadecet to represent codepoints in the BMP, and two hexadecets to represent [UNICODE] codepoints outside the BMP. When stored in nonet pairs, this results in as many as four wasted bits per [UNICODE] character. This transformation format requires complex surrogates to represent codepoints outside the BMP, and can not represent non-[UNICODE] codepoints at all.

[UTF-7]
: requires one to five septets to represent codepoints in the BMP, and as many as eight septets to represent codepoints outside the BMP. When stored in nonets, this results in as many as sixteen wasted bits per character. This transformation format requires very complex and computationally expensive shifting and "modified BASE64" processing, and can not represent non-[UNICODE] codepoints at all.

By comparison, UTF-9 uses one to two nonets to represent codepoints in the BMP, three nonets to represent [UNICODE] codepoints outside the BMP, and three or four nonets to represent non-[UNICODE] codepoints. There are no wasted bits, and as the examples in this document demonstrate, the computational processing is minimal.

Transformation between [UTF-8] and UTF-9 is straightforward, with most of the complexity in the handling of [UTF-8]. It is hoped that future extensions to protocols such as SMTP will permit the use of UTF-9 in these protocols between nonet platforms without the use of [UTF-8] as an "on the wire" format.

Similarly, transformation between [UNICODE] codepoints and UTF-18 is also quite simple. Although (like UCS-2) UTF-18 only represents a subset of the available [UNICODE] codepoints, it encompasses the non-private codepoints that are currently assigned in [UNICODE].

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [KEYWORDS].

2. Overview

UTF-9 encodes [UNICODE] codepoints in the low order 8 bits of a nonet, using the high order bit to indicate continuation. Surrogates are not used.

[UNICODE] codepoints in the range U+0000 - U+00FF ([US-ASCII] and Latin 1) are represented by a single nonet; codepoints in the range U+0100 - U+FFFF (the remainder of the BMP) are represented by two nonets; and codepoints in the range U+1000 - U+10FFFF (remainder of [UNICODE]) are represented by three nonets.

Non-[UNICODE] codepoints in [ISO-10646] (that is, codepoints in the range 0x110000 - 0x7fffffff) can also be represented in UTF-9 by obvious extension, but this is not discussed further as these codepoints have been removed from [ISO-10646] by ISO.

UTF-18 encodes [UNICODE] codepoints in the Basic Multilingual Plane (BMP, plane 0), Supplementary Multilingual Plane (SMP, plane 1), Supplementary Ideographic Plane (SIP, plane 2), and Supplementary Special-purpose Plane (SSP, plane 14) in a single 18-bit value. It does not encode planes 3 though 13, which are currently unused; nor planes 15 or 16, which are private spaces.

Normally, UTF-9 and UTF-18 should only be used in the context of 9 bit storage and transport. Although some protocols, e.g., [FTP], support transport of nonets, the current IETF protocol suite is quite deficient in this area. The IETF is urged to take action to improve IETF protocol support for nonets.

3. UTF-9 Definition

A UTF-9 stream represents [ISO-10646] codepoints using 9 bit nonets. The low order 8-bits of a nonet is an octet, and the high order bit indicates continuation.

UTF-9 does not use surrogates; consequently a UTF-16 value must be transformed into the UCS-4 equivalent, and U+D800 - U+DBFF are never transmitted in UTF-9.

Octets of the [UNICODE] codepoint value are then copied into successive UTF-9 nonets, starting with the most-significant non-zero octet. All but the least significant octet have the continuation bit set in the associated nonet.

Examples:

Character	Name	UTF-9 (in octal)
U+0041	LATIN CAPITAL LETTER A	101
U+00C0	LATIN CAPITAL LETTER A WITH GRAVE	300
U+0391	GREEK CAPITAL LETTER ALPHA	403 221
U+611B	<CJK ideograph meaning "love">	541 33
U+10330	GOTHIC LETTER AHSA	401 403 60
U+E0041	TAG LATIN CAPITAL LETTER A	416 400 101
U+10FFFD	<Plane 16 Private Use, Last>	420 777 375
0x345ecf1b	(UCS-4 value not in [UNICODE])	464 536 717 33

4. UTF-18 Definition

A UTF-18 stream represents [ISO-10646] codepoints using a pair of 9 bit nonets to form an 18-bit value.

UTF-18 does not use surrogates; consequently a UTF-16 value must be transformed into the UCS-4 equivalent, and U+D800 - U+DBFF are never transmitted in UTF-18.

[UNICODE] codepoint values in the range U+0000 - U+2FFFF are copied as the same value into a UTF-18 value. [UNICODE] codepoint values in the range U+E0000 - U+EFFFF are copied as values 0x30000 - 0x3ffff; that is, these values are shifted by 0x70000. Other codepoint values can not be represented in UTF-18.

Examples:

Character	Name	UTF-18 (in octal)
U+0041	LATIN CAPITAL LETTER A	000101
U+00C0	LATIN CAPITAL LETTER A WITH GRAVE	000300
U+0391	GREEK CAPITAL LETTER ALPHA	001621
U+611B	<CJK ideograph meaning "love">	060433
U+10330	GOTHIC LETTER AHSA	201460
U+E0041	TAG LATIN CAPITAL LETTER A	600101

5. Sample Routines

5.1. [UNICODE] Codepoint to UTF-9 Conversion

The following routines demonstrate conversion from UCS-4 to UTF-9. For simplicity, these routines do not do any validity checking. Routines used in applications SHOULD reject invalid UTF-9 sequences; that is, the first nonet with a value of 400 octal (0x100), or sequences that result in an overflow (exceeding 0x10ffff for [UNICODE]), or codepoints used for UTF-16 surrogates.

```
; Return UCS-4 value from UTF-9 string (PDP-10 assembly version)
; Accepts: P1/ 9-bit byte pointer to UTF-9 string
; Returns +1: Always, T1/ UCS-4 value, P1/ updated byte pointer
; Clobbers T2

UT92U4: TDZA T1,T1              ; start with zero
U92U41:  XOR T1,T2              ; insert octet into UCS-4 value
        LSH T1,^D8              ; shift UCS-4 value
        ILDB T2,P1              ; get next nonet
        TRZE T2,400             ; extract octet, any continuation?
         JRST U92U41            ; yes, continue
        XOR T1,T2               ; insert final octet
        POPJ P,
```

```
/* Return UCS-4 value from UTF-9 string (C version)
 * Accepts: pointer to pointer to UTF-9 string
 * Returns: UCS-4 character, nonet pointer updated
 */

UINT31 UTF9_to_UCS4 (UINT9 **utf9PP)
{
  UINT9 nonet;
  UINT31 ucs4;
  for (ucs4 = (nonet = *(*utf9PP)++) & 0xff;
       nonet & 0x100;
       ucs4 |= (nonet = *(*utf9PP)++) & 0xff)
    ucs4 <<= 8;
  return ucs4;
}
```

5.2. UTF-9 to UCS-4 Conversion

The following routines demonstrate conversion from UTF-9 to UCS-4. For simplicity, these routines do not do any validity checking. Routines used in applications SHOULD reject invalid UCS-4 codepoints; that is, codepoints used for UTF-16 surrogates or codepoints with values exceeding 0x10ffff for [UNICODE].

```
; Write UCS-4 character to UTF-9 string (PDP-10 assembly version)
; Accepts: P1/ 9-bit byte pointer to UTF-9 string
;          T1/ UCS-4 character to write
; Returns +1: Always, P1/ updated byte pointer
; Clobbers T1, T2; (T1, T2) must be an accumulator pair

U42UT9: SETO T2,            ; we'll need some of these 1-bits later
        ASHC T1,-^D8        ; low octet becomes nonet with high 0-bit
U32U91: JUMPE T1,U42U9X     ; done if no more octets
        LSHC T1,-^D8        ; shift next octet into T2
        ROT T2,-1           ; turn it into nonet with high 1 bit
        PUSHJ P,U42U91      ; recurse for remainder
U42U9X: LSHC T1,^D9         ; get next nonet back from T2
        IDPB T1,P1          ; write nonet
        POPJ P,

/* Write UCS-4 character to UTF-9 string (C version)
 * Accepts: pointer to nonet string
 *          UCS-4 character to write
 * Returns: updated pointer
 */

UINT9 *UCS4_to_UTF9 (UINT9 *utf9P,UINT31 ucs4)
{
  if (ucs4 > 0x100) {
    if (ucs4 > 0x10000) {
      if (ucs4 > 0x1000000)
        *utf9P++ = 0x100 | ((ucs4 >> 24) & 0xff);
      *utf9P++ = 0x100 | ((ucs4 >> 16) & 0xff);
    }
    *utf9P++ = 0x100 | ((ucs4 >> 8) & 0xff);
  }
  *utf9P++ = ucs4 & 0xff;
  return utf9P;
}
```

6. Implementation Experience

As the sample routines demonstrate, it is quite simple to implement UTF-9 and UTF-18 on a nonet-based architecture. More sophisticated routines can be found in ftp://panda.com/tops-20/utools.mac.txt or from lingling.panda.com via the file <UTF9>UTOOLS.MAC via ANONYMOUS [FTP].

We are now in the process of implementing support for nonet-based text files and automated transformation between septet, octet, and nonet textual data.

7. References

7.1. Normative References

[FTP] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[IAB-CHARACTER] Weider, C., Preston, C., Simonsen, K., Alvestrand, H., Atkinson, R., Crispin, M., and P. Svanberg, "The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996", RFC 2130, April 1997.

[ISO-10646] International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set (UCS)", ISO/IEC Standard 10646, comprised of ISO/IEC 10646-1:2000, "Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-2:2001, "Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 2: Supplementary Planes" and ISO/IEC 10646-1:2000/Amd 1:2002, "Mathematical symbols and other characters".

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[UNICODE] The Unicode Consortium, "The Unicode Standard - Version 3.2", defined by The Unicode Standard, Version 3.0 (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the Unicode Standard Annex #27: Unicode 3.1 and by the Unicode Standard Annex #28: Unicode 3.2, March 2002.

7.2. Informative References

[US-ASCII] American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[UTF-16] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[UTF-7] Goldsmith, D. and M. Davis, "UTF-7 A Mail-Safe Transformation Format of Unicode", RFC 2152, May 1997.

[UTF-8] Sollins, K., "Architectural Principles of Uniform Resource Name Resolution", RFC 2276, January 1998.

8. Security Considerations

As with UTF-8, UTF-9 can represent codepoints that are not in [UNICODE]. Applications should validate UTF-9 strings to ensure that all codepoints do not exceed the [UNICODE] maximum of U+10FFFF.

The sample routines in this document are for example purposes, and make no attempt to validate their arguments, e.g., test for overflow ([UNICODE] values great than 0x10ffff) or codepoints used for surrogates. Besides resulting in invalid data, this can also create covert channels.

9. IANA Considerations

The IANA shall reserve the charset names "UTF-9" and "UTF-18" for future assignment.

Author's Address

Mark R. Crispin
Panda Programming
6158 NE Lariat Loop
Bainbridge Island, WA 98110-2098

Phone: (206) 842-2385
EMail: UTF9@Lingling.Panda.COM

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.