Network Working Group
Request for Comments: 2279
Obsoletes: 2044
Category: Standards Track

F. Yergeau
Alis Technologies

January 1998

UTF-8, a transformation format of ISO 10646

Status of this Memo

Copyright Notice

Abstract

ISO/IEC 10646-1 defines a multi-octet character set called the Universal Character Set (UCS) which encompasses most of the world's writing systems. Multi-octet characters, however, are not compatible with many current applications and protocols, and this has led to the development of a few so-called UCS transformation formats (UTF), each with different characteristics. UTF-8, the object of this memo, has the characteristic of preserving the full US-ASCII range, providing compatibility with file systems, parsers and other software that rely on US-ASCII values but are transparent to other values. This memo updates and replaces RFC 2044, in particular addressing the question of versions of the relevant standards.

1. Introduction

ISO/IEC 10646-1 [ISO-10646] defines a multi-octet character set called the Universal Character Set (UCS), which encompasses most of the world's writing systems. Two multi-octet encodings are defined, a four-octet per character encoding called UCS-4 and a two-octet per character encoding called UCS-2, able to address only the first 64K characters of the UCS (the Basic Multilingual Plane, BMP), outside of which there are currently no assignments.

It is noteworthy that the same set of characters is defined by the Unicode standard [UNICODE], which further defines additional character properties and other application details of great interest to implementors, but does not have the UCS-4 encoding. Up to the

present time, changes in Unicode and amendments to ISO/IEC 10646 have tracked each other, so that the character repertoires and code point assignments have remained in sync. The relevant standardization committees have committed to maintain this very useful synchronism.

The UCS-2 and UCS-4 encodings, however, are hard to use in many current applications and protocols that assume 8 or even 7 bit characters. Even newer systems able to deal with 16 bit characters cannot process UCS-4 data. This situation has led to the development of so-called UCS transformation formats (UTF), each with different characteristics.

UTF-1 has only historical interest, having been removed from ISO/IEC 10646. UTF-7 has the quality of encoding the full BMP repertoire using only octets with the high-order bit clear (7 bit US-ASCII values, [US-ASCII]), and is thus deemed a mail-safe encoding ([RFC2152]). UTF-8, the object of this memo, uses all bits of an octet, but has the quality of preserving the full US-ASCII range: US-ASCII characters are encoded in one octet having the normal US-ASCII value, and any octet with such a value can only stand for an US-ASCII character, and nothing else.

UTF-16 is a scheme for transforming a subset of the UCS-4 repertoire into pairs of UCS-2 values from a reserved range. UTF-16 impacts UTF-8 in that UCS-2 values from the reserved range must be treated specially in the UTF-8 transformation.

UTF-8 encodes UCS-2 or UCS-4 characters as a varying number of octets, where the number of octets, and the value of each, depend on the integer value assigned to the character in ISO/IEC 10646. This transformation format has the following characteristics (all values are in hexadecimal):

- Character values from 0000 0000 to 0000 007F (US-ASCII repertoire) correspond to octets 00 to 7F (7 bit US-ASCII values). A direct consequence is that a plain ASCII string is also a valid UTF-8 string.

- US-ASCII values do not appear otherwise in a UTF-8 encoded character stream. This provides compatibility with file systems or other software (e.g. the printf() function in C libraries) that parse based on US-ASCII values but are transparent to other values.

- Round-trip conversion is easy between UTF-8 and either of UCS-4, UCS-2.

- The first octet of a multi-octet sequence indicates the number of octets in the sequence.

- The octet values FE and FF never appear.

- Character boundaries are easily found from anywhere in an octet stream.

- The lexicographic sorting order of UCS-4 strings is preserved. Of course this is of limited interest since the sort order is not culturally valid in either case.

- The Boyer-Moore fast search algorithm can be used with UTF-8 data.

- UTF-8 strings can be fairly reliably recognized as such by a simple algorithm, i.e. the probability that a string of characters in any other encoding appears as valid UTF-8 is low, diminishing with increasing string length.

UTF-8 was originally a project of the X/Open Joint Internationalization Group XOJIG with the objective to specify a File System Safe UCS Transformation Format [FSS-UTF] that is compatible with UNIX systems, supporting multilingual text in a single encoding. The original authors were Gary Miller, Greger Leijonhufvud and John Entenmann. Later, Ken Thompson and Rob Pike did significant work for the formal UTF-8.

A description can also be found in Unicode Technical Report #4 and in the Unicode Standard, version 2.0 [UNICODE]. The definitive reference, including provisions for UTF-16 data within UTF-8, is Annex R of ISO/IEC 10646-1 [ISO-10646].

2. UTF-8 definition

In UTF-8, characters are encoded using sequences of 1 to 6 octets. The only octet of a "sequence" of one has the higher-order bit set to 0, the remaining 7 bits being used to encode the character value. In a sequence of n octets, n>1, the initial octet has the n higher-order bits set to 1, followed by a bit set to 0. The remaining bit(s) of that octet contain bits from the value of the character to be encoded. The following octet(s) all have the higher-order bit set to 1 and the following bit set to 0, leaving 6 bits in each to contain bits from the character to be encoded.

The table below summarizes the format of these different octet types. The letter x indicates bits available for encoding bits of the UCS-4 character value.

```
UCS-4 range (hex.)           UTF-8 octet sequence (binary)
0000 0000-0000 007F   0xxxxxxx
0000 0080-0000 07FF   110xxxxx 10xxxxxx
0000 0800-0000 FFFF   1110xxxx 10xxxxxx 10xxxxxx

0001 0000-001F FFFF   11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0020 0000-03FF FFFF   111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0400 0000-7FFF FFFF   1111110x 10xxxxxx ... 10xxxxxx
```

Encoding from UCS-4 to UTF-8 proceeds as follows:

1) Determine the number of octets required from the character value and the first column of the table above. It is important to note that the rows of the table are mutually exclusive, i.e. there is only one valid way to encode a given UCS-4 character.

2) Prepare the high-order bits of the octets as per the second column of the table.

3) Fill in the bits marked x from the bits of the character value, starting from the lower-order bits of the character value and putting them first in the last octet of the sequence, then the next to last, etc. until all x bits are filled in.

The algorithm for encoding UCS-2 (or Unicode) to UTF-8 can be obtained from the above, in principle, by simply extending each UCS-2 character with two zero-valued octets. However, pairs of UCS-2 values between D800 and DFFF (surrogate pairs in Unicode parlance), being actually UCS-4 characters transformed through UTF-16, need special treatment: the UTF-16 transformation must be undone, yielding a UCS-4 character that is then transformed as above.

Decoding from UTF-8 to UCS-4 proceeds as follows:

1) Initialize the 4 octets of the UCS-4 character with all bits set to 0.

2) Determine which bits encode the character value from the number of octets in the sequence and the second column of the table above (the bits marked x).

3) Distribute the bits from the sequence to the UCS-4 character, first the lower-order bits from the last octet of the sequence and proceeding to the left until no x bits are left.

If the UTF-8 sequence is no more than three octets long, decoding can proceed directly to UCS-2.

> NOTE -- actual implementations of the decoding algorithm above should protect against decoding invalid sequences. For instance, a naive implementation may (wrongly) decode the invalid UTF-8 sequence C0 80 into the character U+0000, which may have security consequences and/or cause other problems. See the Security Considerations section below.

A more detailed algorithm and formulae can be found in [FSS_UTF], [UNICODE] or Annex R to [ISO-10646].

3. Versions of the standards

ISO/IEC 10646 is updated from time to time by published amendments; similarly, different versions of the Unicode standard exist: 1.0, 1.1 and 2.0 as of this writing. Each new version obsoletes and replaces the previous one, but implementations, and more significantly data, are not updated instantly.

In general, the changes amount to adding new characters, which does not pose particular problems with old data. Amendment 5 to ISO/IEC 10646, however, has moved and expanded the Korean Hangul block, thereby making any previous data containing Hangul characters invalid under the new version. Unicode 2.0 has the same difference from Unicode 1.1. The official justification for allowing such an incompatible change was that no implementations and no data containing Hangul existed, a statement that is likely to be true but remains unprovable. The incident has been dubbed the "Korean mess", and the relevant committees have pledged to never, ever again make such an incompatible change.

New versions, and in particular any incompatible changes, have q conseuences regarding MIME character encoding labels, to be discussed in section 5.

4. Examples

The UCS-2 sequence "A<NOT IDENTICAL TO><ALPHA>." (0041, 2262, 0391, 002E) may be encoded in UTF-8 as follows:

41 E2 89 A2 CE 91 2E

The UCS-2 sequence representing the Hangul characters for the Korean word "hangugo" (D55C, AD6D, C5B4) may be encoded as follows:

ED 95 9C EA B5 AD EC 96 B4

The UCS-2 sequence representing the Han characters for the Japanese word "nihongo" (65E5, 672C, 8A9E) may be encoded as follows:

E6 97 A5 E6 9C AC E8 AA 9E

5. MIME registration

This memo is meant to serve as the basis for registration of a MIME character set parameter (charset) [CHARSET-REG]. The proposed charset parameter value is "UTF-8". This string labels media types containing text consisting of characters from the repertoire of ISO/IEC 10646 including all amendments at least up to amendment 5 (Korean block), encoded to a sequence of octets using the encoding scheme outlined above. UTF-8 is suitable for use in MIME content types under the "text" top-level type.

It is noteworthy that the label "UTF-8" does not contain a version identification, referring generically to ISO/IEC 10646. This is intentional, the rationale being as follows:

A MIME charset label is designed to give just the information needed to interpret a sequence of bytes received on the wire into a sequence of characters, nothing more (see RFC 2045, section 2.2, in [MIME]). As long as a character set standard does not change incompatibly, version numbers serve no purpose, because one gains nothing by learning from the tag that newly assigned characters may be received that one doesn't know about. The tag itself doesn't teach anything about the new characters, which are going to be received anyway.

Hence, as long as the standards evolve compatibly, the apparent advantage of having labels that identify the versions is only that, apparent. But there is a disadvantage to such version-dependent labels: when an older application receives data accompanied by a newer, unknown label, it may fail to recognize the label and be completely unable to deal with the data, whereas a generic, known label would have triggered mostly correct processing of the data, which may well not contain any new characters.

Now the "Korean mess" (ISO/IEC 10646 amendment 5) is an incompatible change, in principle contradicting the appropriateness of a version independent MIME charset label as described above. But the compatibility problem can only appear with data containing Korean Hangul characters encoded according to Unicode 1.1 (or equivalently ISO/IEC 10646 before amendment 5), and there is arguably no such data to worry about, this being the very reason the incompatible change was deemed acceptable.

In practice, then, a version-independent label is warranted, provided the label is understood to refer to all versions after Amendment 5, and provided no incompatible change actually occurs. Should incompatible changes occur in a later version of ISO/IEC 10646, the MIME charset label defined here will stay aligned with the previous version until and unless the IETF specifically decides otherwise.

It is also proposed to register the charset parameter value "UNICODE-1-1-UTF-8", for the exclusive purpose of labelling text data containing Hangul syllables encoded to UTF-8 without taking into account Amendment 5 of ISO/IEC 10646 (i.e. using the pre-amendment 5 code point assignments). Any other UTF-8 data SHOULD NOT use this label, in particular data not containing any Hangul syllables, and it is felt important to strongly recommend against creating any new Hangul-containing data without taking Amendment 5 of ISO/IEC 10646 into account.

6. Security Considerations

Implementors of UTF-8 need to consider the security aspects of how they handle illegal UTF-8 sequences. It is conceivable that in some circumstances an attacker would be able to exploit an incautious UTF-8 parser by sending it an octet sequence that is not permitted by the UTF-8 syntax.

A particularly subtle form of this attack could be carried out against a parser which performs security-critical validity checks against the UTF-8 encoded form of its input, but interprets certain illegal octet sequences as characters. For example, a parser might prohibit the NUL character when encoded as the single-octet sequence 00, but allow the illegal two-octet sequence C0 80 and interpret it as a NUL character. Another example might be a parser which prohibits the octet sequence 2F 2E 2E 2F ("/../"), yet permits the illegal octet sequence 2F C0 AE 2E 2F.

Acknowledgments

The following have participated in the drafting and discussion of this memo:

James E. Agenbroad	Andries Brouwer
Martin J. D|rst	Ned Freed
David Goldsmith	Edwin F. Hart
Kent Karlsson	Markus Kuhn
Michael Kung	Alain LaBonte
John Gardiner Myers	Murray Sargent
Keld Simonsen	Arnold Winkler

Bibliography

[CHARSET-REG] Freed, N., and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2278, January 1998.

[FSS_UTF] X/Open CAE Specification C501 ISBN 1-85912-082-2 28cm. 22p. pbk. 172g. 4/95, X/Open Company Ltd., "File System Safe UCS Transformation Format (FSS_UTF)", X/Open Preleminary Specification, Document Number P316. Also published in Unicode Technical Report #4.

[ISO-10646] 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-8 is described in Annex R, published as Amendment 2. UTF-16 is described in Annex Q, published as Amendment 1. 17 other amendments are currently at various stages of standardization.

[MIME] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045. N. Freed, N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046. K. Moore, "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047. N. Freed, J. Klensin, J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", RFC 2048. N. Freed, N. Borenstein, " Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049. All November 1996.

[RFC2152] Goldsmith, D., and M. Davis, "UTF-7: A Mail-safe Transformation Format of Unicode", RFC 1642, Taligent inc., May 1997. (Obsoletes RFC1642)

[UNICODE] The Unicode Consortium, "The Unicode Standard -- Version 2.0", Addison-Wesley, 1996.

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

Author's Address

Francois Yergeau
Alis Technologies
100, boul. Alexis-Nihon
Suite 600
Montreal QC H4M 2P2
Canada

Phone: +1 (514) 747-2547
Fax: +1 (514) 747-2561
EMail: fyergeau@alis.com

Full Copyright Statement