Network Working Group
Request for Comments: 3629
STD: 63
Obsoletes: 2279
Category: Standards Track

F. Yergeau
Alis Technologies

November 2003

UTF-8, a transformation format of ISO 10646

Status of this Memo

Copyright Notice

Abstract

ISO/IEC 10646-1 defines a large character set called the Universal Character Set (UCS) which encompasses most of the world's writing systems. The originally proposed encodings of the UCS, however, were not compatible with many current applications and protocols, and this has led to the development of UTF-8, the object of this memo. UTF-8 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 obsoletes and replaces RFC 2279.

Table of Contents

1. Introduction

ISO/IEC 10646 [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 characters is defined by the Unicode standard [UNICODE], which further defines additional character properties and other application details of great interest to implementers. Up to the present time, changes in Unicode and amendments and additions 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.

ISO/IEC 10646 and Unicode define several encoding forms of their common repertoire: UTF-8, UCS-2, UTF-16, UCS-4 and UTF-32. In an encoding form, each character is represented as one or more encoding units. All standard UCS encoding forms except UTF-8 have an encoding unit larger than one octet, making them hard to use in many current applications and protocols that assume 8 or even 7 bit characters.

UTF-8, the object of this memo, has a one-octet encoding unit. It uses all bits of an octet, but has the quality of preserving the full US-ASCII [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 a US-ASCII character, and nothing else.

UTF-8 encodes UCS 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 (the character number, a.k.a. code position, code point or Unicode scalar value). This encoding form has the following characteristics (all values are in hexadecimal):

- Character numbers from U+0000 to U+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 octet 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 other encoding forms.

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

- The octet values C0, C1, F5 to FF never appear.

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

- The byte-value lexicographic sorting order of UTF-8 strings is the same as if ordered by character numbers. Of course this is of limited interest since a sort order based on character numbers is almost never culturally valid.

- 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 devised in September 1992 by Ken Thompson, guided by design criteria specified by Rob Pike, with the objective of defining a UCS transformation format usable in the Plan9 operating system in a non-disruptive manner. Thompson's design was stewarded through standardization by the X/Open Joint Internationalization Group XOJIG (see [FSS_UTF]), bearing the names FSS-UTF (variant FSS/UTF), UTF-2 and finally UTF-8 along the way.

2. Notational conventions

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 [RFC2119].

UCS characters are designated by the U+HHHH notation, where HHHH is a string of from 4 to 6 hexadecimal digits representing the character number in ISO/IEC 10646.

3. UTF-8 definition

UTF-8 is defined by the Unicode Standard [UNICODE]. Descriptions and formulae can also be found in Annex D of ISO/IEC 10646-1 [ISO.10646]

In UTF-8, characters from the U+0000..U+10FFFF range (the UTF-16 accessible range) are encoded using sequences of 1 to 4 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 number. 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 number 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 character number.

Char. number range (hexadecimal)	UTF-8 octet sequence (binary)
0000 0000-0000 007F	0xxxxxxx
0000 0080-0000 07FF	110xxxxx 10xxxxxx
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Encoding a character to UTF-8 proceeds as follows:

1. Determine the number of octets required from the character number 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 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 number, expressed in binary. Start by putting the lowest-order bit of the character number in the lowest-order position of the last octet of the sequence, then put the next higher-order bit of the character number in the next higher-order position of that octet, etc. When the x bits of the last octet are filled in, move on to the next to last octet, then to the preceding one, etc. until all x bits are filled in.

The definition of UTF-8 prohibits encoding character numbers between U+D800 and U+DFFF, which are reserved for use with the UTF-16 encoding form (as surrogate pairs) and do not directly represent characters. When encoding in UTF-8 from UTF-16 data, it is necessary to first decode the UTF-16 data to obtain character numbers, which are then encoded in UTF-8 as described above. This contrasts with CESU-8 [CESU-8], which is a UTF-8-like encoding that is not meant for use on the Internet. CESU-8 operates similarly to UTF-8 but encodes the UTF-16 code values (16-bit quantities) instead of the character number (code point). This leads to different results for character numbers above 0xFFFF; the CESU-8 encoding of those characters is NOT valid UTF-8.

Decoding a UTF-8 character proceeds as follows:

1. Initialize a binary number with all bits set to 0. Up to 21 bits may be needed.

2. Determine which bits encode the character number 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 binary number, first the lower-order bits from the last octet of the sequence and proceeding to the left until no x bits are left. The binary number is now equal to the character number.

Implementations of the decoding algorithm above MUST protect against decoding invalid sequences. For instance, a naive implementation may decode the overlong UTF-8 sequence C0 80 into the character U+0000, or the surrogate pair ED A1 8C ED BE B4 into U+233B4. Decoding invalid sequences may have security consequences or cause other problems. See Security Considerations (Section 10) below.

4. Syntax of UTF-8 Byte Sequences

For the convenience of implementors using ABNF, a definition of UTF-8 in ABNF syntax is given here.

A UTF-8 string is a sequence of octets representing a sequence of UCS characters. An octet sequence is valid UTF-8 only if it matches the following syntax, which is derived from the rules for encoding UTF-8 and is expressed in the ABNF of [RFC2234].

```
UTF8-octets = *( UTF8-char )
UTF8-char   = UTF8-1 / UTF8-2 / UTF8-3 / UTF8-4
UTF8-1      = %x00-7F
UTF8-2      = %xC2-DF UTF8-tail
```

```
UTF8-3      = %xE0 %xA0-BF UTF8-tail / %xE1-EC 2( UTF8-tail ) /
              %xED %x80-9F UTF8-tail / %xEE-EF 2( UTF8-tail )
UTF8-4      = %xF0 %x90-BF 2( UTF8-tail ) / %xF1-F3 3( UTF8-tail ) /
              %xF4 %x80-8F 2( UTF8-tail )
UTF8-tail   = %x80-BF
```

NOTE -- The authoritative definition of UTF-8 is in [UNICODE]. This grammar is believed to describe the same thing Unicode describes, but does not claim to be authoritative. Implementors are urged to rely on the authoritative source, rather than on this ABNF.

5. Versions of the standards

ISO/IEC 10646 is updated from time to time by publication of amendments and additional parts; similarly, new versions of the Unicode standard are published over time. 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. In 1996, Amendment 5 to the 1993 edition of ISO/IEC 10646 and Unicode 2.0 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 justification for allowing such an incompatible change was that there were no major implementations and no significant amounts of data containing Hangul. The incident has been dubbed the "Korean mess", and the relevant committees have pledged to never, ever again make such an incompatible change (see Unicode Consortium Policies [1]).

New versions, and in particular any incompatible changes, have consequences regarding MIME charset labels, to be discussed in MIME registration (Section 8).

6. Byte order mark (BOM)

The UCS character U+FEFF "ZERO WIDTH NO-BREAK SPACE" is also known informally as "BYTE ORDER MARK" (abbreviated "BOM"). This character can be used as a genuine "ZERO WIDTH NO-BREAK SPACE" within text, but the BOM name hints at a second possible usage of the character: to prepend a U+FEFF character to a stream of UCS characters as a "signature". A receiver of such a serialized stream may then use the initial character as a hint that the stream consists of UCS characters and also to recognize which UCS encoding is involved and, with encodings having a multi-octet encoding unit, as a way to

recognize the serialization order of the octets. UTF-8 having a single-octet encoding unit, this last function is useless and the BOM will always appear as the octet sequence EF BB BF.

It is important to understand that the character U+FEFF appearing at any position other than the beginning of a stream MUST be interpreted with the semantics for the zero-width non-breaking space, and MUST NOT be interpreted as a signature. When interpreted as a signature, the Unicode standard suggests than an initial U+FEFF character may be stripped before processing the text. Such stripping is necessary in some cases (e.g., when concatenating two strings, because otherwise the resulting string may contain an unintended "ZERO WIDTH NO-BREAK SPACE" at the connection point), but might affect an external process at a different layer (such as a digital signature or a count of the characters) that is relying on the presence of all characters in the stream. It is therefore RECOMMENDED to avoid stripping an initial U+FEFF interpreted as a signature without a good reason, to ignore it instead of stripping it when appropriate (such as for display) and to strip it only when really necessary.

U+FEFF in the first position of a stream MAY be interpreted as a zero-width non-breaking space, and is not always a signature. In an attempt at diminishing this uncertainty, Unicode 3.2 adds a new character, U+2060 "WORD JOINER", with exactly the same semantics and usage as U+FEFF except for the signature function, and strongly recommends its exclusive use for expressing word-joining semantics. Eventually, following this recommendation will make it all but certain that any initial U+FEFF is a signature, not an intended "ZERO WIDTH NO-BREAK SPACE".

In the meantime, the uncertainty unfortunately remains and may affect Internet protocols. Protocol specifications MAY restrict usage of U+FEFF as a signature in order to reduce or eliminate the potential ill effects of this uncertainty. In the interest of striking a balance between the advantages (reduction of uncertainty) and drawbacks (loss of the signature function) of such restrictions, it is useful to distinguish a few cases:

- A protocol SHOULD forbid use of U+FEFF as a signature for those textual protocol elements that the protocol mandates to be always UTF-8, the signature function being totally useless in those cases.

- A protocol SHOULD also forbid use of U+FEFF as a signature for those textual protocol elements for which the protocol provides character encoding identification mechanisms, when it is expected that implementations of the protocol will be in a position to always use the mechanisms properly. This will be the case when

the protocol elements are maintained tightly under the control of the implementation from the time of their creation to the time of their (properly labeled) transmission.

- o A protocol SHOULD NOT forbid use of U+FEFF as a signature for those textual protocol elements for which the protocol does not provide character encoding identification mechanisms, when a ban would be unenforceable, or when it is expected that implementations of the protocol will not be in a position to always use the mechanisms properly. The latter two cases are likely to occur with larger protocol elements such as MIME entities, especially when implementations of the protocol will obtain such entities from file systems, from protocols that do not have encoding identification mechanisms for payloads (such as FTP) or from other protocols that do not guarantee proper identification of character encoding (such as HTTP).

When a protocol forbids use of U+FEFF as a signature for a certain protocol element, then any initial U+FEFF in that protocol element MUST be interpreted as a "ZERO WIDTH NO-BREAK SPACE". When a protocol does NOT forbid use of U+FEFF as a signature for a certain protocol element, then implementations SHOULD be prepared to handle a signature in that element and react appropriately: using the signature to identify the character encoding as necessary and stripping or ignoring the signature as appropriate.

7. Examples

The character sequence U+0041 U+2262 U+0391 U+002E "A<NOT IDENTICAL TO><ALPHA>." is encoded in UTF-8 as follows:

```
--+--------+-----+--
41 E2 89 A2 CE 91 2E
--+--------+-----+--
```

The character sequence U+D55C U+AD6D U+C5B4 (Korean "hangugeo", meaning "the Korean language") is encoded in UTF-8 as follows:

```
--------+--------+--------
ED 95 9C EA B5 AD EC 96 B4
--------+--------+--------
```

The character sequence U+65E5 U+672C U+8A9E (Japanese "nihongo", meaning "the Japanese language") is encoded in UTF-8 as follows:

```
--------+--------+--------
E6 97 A5 E6 9C AC E8 AA 9E
--------+--------+--------
```

The character U+233B4 (a Chinese character meaning 'stump of tree'), prepended with a UTF-8 BOM, is encoded in UTF-8 as follows:

```
--------+-----------
EF BB BF F0 A3 8E B4
--------+-----------
```

8. MIME registration

This memo serves as the basis for registration of the MIME charset parameter for UTF-8, according to [RFC2978]. The 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 of the 1993 edition (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 [RFC2045], section 2.2). 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.

9. IANA Considerations

The entry for UTF-8 in the IANA charset registry has been updated to point to this memo.

10. Security Considerations

Implementers 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 can 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 erroneously 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. This last exploit has actually been used in a widespread virus attacking Web servers in 2001; thus, the security threat is very real.

Another security issue occurs when encoding to UTF-8: the ISO/IEC 10646 description of UTF-8 allows encoding character numbers up to U+7FFFFFFF, yielding sequences of up to 6 bytes. There is therefore a risk of buffer overflow if the range of character numbers is not explicitly limited to U+10FFFF or if buffer sizing doesn’t take into account the possibility of 5- and 6-byte sequences.

Security may also be impacted by a characteristic of several character encodings, including UTF-8: the "same thing" (as far as a user can tell) can be represented by several distinct character sequences. For instance, an e with acute accent can be represented by the precomposed U+00E9 E ACUTE character or by the canonically equivalent sequence U+0065 U+0301 (E + COMBINING ACUTE). Even though UTF-8 provides a single byte sequence for each character sequence, the existence of multiple character sequences for "the same thing" may have security consequences whenever string matching, indexing,

searching, sorting, regular expression matching and selection are involved. An example would be string matching of an identifier appearing in a credential and in access control list entries. This issue is amenable to solutions based on Unicode Normalization Forms, see [UAX15].

11. Acknowledgements

The following have participated in the drafting and discussion of this memo: James E. Agenbroad, Harald Alvestrand, Andries Brouwer, Mark Davis, Martin J. Duerst, Patrick Faltstrom, Ned Freed, David Goldsmith, Tony Hansen, Edwin F. Hart, Paul Hoffman, David Hopwood, Simon Josefsson, Kent Karlsson, Dan Kohn, Markus Kuhn, Michael Kung, Alain LaBonte, Ira McDonald, Alexey Melnikov, MURATA Makoto, John Gardiner Myers, Chris Newman, Dan Oscarsson, Roozbeh Pournader, Murray Sargent, Markus Scherer, Keld Simonsen, Arnold Winkler, Kenneth Whistler and Misha Wolf.

12. Changes from RFC 2279

- Restricted the range of characters to 0000-10FFFF (the UTF-16 accessible range).
- Made Unicode the source of the normative definition of UTF-8, keeping ISO/IEC 10646 as the reference for characters.
- Straightened out terminology. UTF-8 now described in terms of an encoding form of the character number. UCS-2 and UCS-4 almost disappeared.
- Turned the note warning against decoding of invalid sequences into a normative MUST NOT.
- Added a new section about the UTF-8 BOM, with advice for protocols.
- Removed suggested UNICODE-1-1-UTF-8 MIME charset registration.
- Added an ABNF syntax for valid UTF-8 octet sequences
- Expanded Security Considerations section, in particular impact of Unicode normalization

13. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 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".

[UNICODE] The Unicode Consortium, "The Unicode Standard -- Version 4.0", defined by The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1), April 2003, <http://www.unicode.org/unicode/standard/versions/enumeratedversions.html#Unicode_4_0_0>.

14. Informative References

[CESU-8] Phipps, T., "Unicode Technical Report #26: Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8)", UTR 26, April 2002, <http://www.unicode.org/unicode/reports/tr26/>.

[FSS_UTF] X/Open Company Ltd., "X/Open Preliminary Specification -- File System Safe UCS Transformation Format (FSS-UTF)", May 1993, <http://wwwold.dkuug.dk/jtc1/sc22/wg20/docs/N193-FSS-UTF.pdf>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, October 2000.

[UAX15] Davis, M. and M. Duerst, "Unicode Standard Annex #15: Unicode Normalization Forms", An integral part of The Unicode Standard, Version 4.0.0, April 2003, <http://www.unicode.org/unicode/reports/tr15>.

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

15. URIs

[1] <http://www.unicode.org/unicode/standard/policies.html>

16. Intellectual Property Statement

17. Author's Address

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

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

18. Full Copyright Statement

Acknowledgement

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