Network Working Group
Internet-Draft
Expires: March 2, 2003

F. Yergeau
Alis Technologies
September 1, 2002

UTF-8, a transformation format of ISO 10646
draft-yergeau-rfc2279bis-01

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.

This Internet-Draft will expire on March 2, 2003.

Copyright Notice

Copyright (C) The Internet Society (2002). All Rights Reserved.

Abstract

<1>
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 updates and replaces RFC 2279.
<2>
Discussion of this draft should take place on the ietf-

charsets@iana.org mailing list.

Table of Contents

1. Introduction

<3>
ISO/IEC 10646 [ISO.10646-1] 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 implementors. 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.

<4>
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.

<5>
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 an US-ASCII character, and nothing else.

<6>
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 point or Unicode scalar value). This encoding form has the following characteristics (all values are in hexadecimal):

<7>
- 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.

<8>

- 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.

<9>

- Round-trip conversion is easy between UTF-8 and other encoding forms.

<10>

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

<11>

- The octet values C0, C1, FE and FF never appear. If the range of character numbers is restricted to U+0000..U+10FFFF (the UTF-16 accessible range), then the octet values F5..FD also never appear.

<12>

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

<13>

- The 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 not culturally valid.

<14>

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

<15>

- 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.

<16>

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 definition of UTF-8.

2. Notational conventions

<17>
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].

<18>
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

<19>
UTF-8 is defined by Annex D of ISO/IEC 10646-1 [ISO.10646-1]. Descriptions and formulae can also be found in the Unicode Standard [UNICODE] and in [FSS_UTF].

<20>
In UTF-8, characters are encoded using sequences of 1 to 6 octets. If the range of character numbers is restricted to U+0000..U+10FFFF (the UTF-16 accessible range), then only sequences of one to four octets will occur. 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.

<21>
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 0000-001F FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0020 0000-03FF FFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0400 0000-7FFF FFFF	1111110x 10xxxxxx ... 10xxxxxx

<22>
Encoding a character to UTF-8 proceeds as follows:

<23>
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.

<24>
2. Prepare the high-order bits of the octets as per the second column of the table.

<25>
3. Fill in the bits marked x from the bits of the character number, expressed in binary. Start from the lower-order bits of the character number and put them first in the last octet of the sequence, then the next to last, etc. until all x bits are filled in.

<26>
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.

<27>
Decoding a UTF-8 character proceeds as follows:
<28>
1. Initialize a binary number with all bits set to 0. Up to 31 bits may be needed (up to 21 if the range of character numbers is known to be restricted to the UTF-16 accessible range).
<29>
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).
<30>
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.

<31>
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

<32>

A UTF-8 string is a sequence of bytes representing a sequence of UCS characters. The byte 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-string = *( UTF8-char )
UTF8-char   = UTF8-1 /
              UTF8-2-head 1( UTF8-tail ) /
              UTF8-3-head 1( UTF8-tail ) /
              UTF8-4-head 2( UTF8-tail ) /
              UTF8-5-head 3( UTF8-tail ) /
              UTF8-6-head 4( UTF8-tail )
UTF8-1      = %x00-7F
UTF8-2-head = %xC2-DF
UTF8-3-head = %xE0 %xA0-BF / %xE1-EC %x80-BF /
              %xED %x80-9F / %xEE-EF %x80-BF
UTF8-4-head = %xF0 %x90-BF / %xF1-F7 %x80-BF
UTF8-5-head = %xF8 %x88-BF / %xF9-FB %x80-BF
UTF8-6-head = %xFC %x84-BF / %xFD    %x80-BF
UTF8-tail   = %x80-BF

UTF8-string = *( UTF8-char )
UTF8-char   = UTF8-1 / UTF8-2 / UTF8-3 / UTF8-4 / UTF8-5 / UTF8-6
UTF8-char   = UTF8-1 /
              UTF8-2 /
              UTF8-3 /
              UTF8-4 /
              UTF8-5 /
              UTF8-6
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-F7 3( UTF8-tail )
UTF8-5      = %xF8 %x88-BF 3( UTF8-tail ) / %xF9-FB 4( UTF8-tail )
UTF8-6      = %xFC %x84-BF 4( UTF8-tail ) / %xFD 5( UTF8-tail )
UTF8-tail   = %x80-BF
```

5. Versions of the standards

<33>
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.

<34>
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]).

<35>
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)

<36>
The Unicode Standard and ISO 10646 define the character "ZERO WIDTH NO-BREAK SPACE" (U+FEFF), which is also known informally as "BYTE ORDER MARK" (abbreviated "BOM"). The latter name hints at a second possible usage of the character, in addition to its normal use as a genuine "ZERO WIDTH NO-BREAK SPACE" within text. This usage, suggested by Unicode section 2.7 and ISO/IEC 10646 Annex H (informative), is 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. The signature can also be used 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.

<37>
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 byte-order mark. The contrapositive of that statement is not always true: the character U+FEFF in the first position of a stream MAY be interpreted as a zero-width non-breaking space, and is not always a byte-order mark. For example, if a process splits a UCS string into many parts, a part might begin with U+FEFF because there was a zero-width no-break space at the beginning of that substring.

<38>
The Unicode standard further suggests than an initial U+FEFF character may be stripped before processing the text, the rationale being that such a character in initial position may be an artifact of the encoding (an encoding signature), not a genuine intended "ZERO WIDTH NO-BREAK SPACE". Note that such stripping 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.

<39>

In particular, in UTF-8 plain text it is likely, but not certain, that an initial octet sequence of EF BB BF is a signature. When concatenating two strings, it is important to strip out those signatures, because otherwise the resulting string may contain an unintended "ZERO WIDTH NO-BREAK SPACE" at the connection point.

<40>

In an attempt at diminishing the 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".

7. Examples

<41>

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
--+--------+-----+--
```

<42>

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
--------+--------+--------
```

<43>

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
--------+--------+--------
```

<44>
   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

<45>
   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.
<46>
   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:
<47>

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.

<48>

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.

<49>

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.

<50>

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

<51>

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

10. Security Considerations

<52>
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.

<53>
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; the security threat is thus very real.

Bibliography

[FSS_UTF] X/Open Company Ltd., "X/Open CAE Specification C501 -- File System Safe UCS Transformation Format (FSS_UTF)", ISBN 1-85912-082-2, April 1995.

[ISO.10646-1] International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane", ISO Standard 10646-1, 2000.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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.

[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 (see http://www.unicode.org/reports/tr27) and by the Unicode Standard Annex #28: Unicode 3.2 (see http://www.unicode.org/reports/tr28), March 2002, <http://www.unicode.org/unicode/standard/versions/enumeratedversions.html#Unicode_3_2_0>.

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

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

Author's Address

FranÈºois Yergeau
Alis Technologies
100, boul. Alexis-Nihon, bureau 600
MontrÈ¨al, QC H4M 2P2
Canada

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

Appendix A. Acknowledgements

<62>

The following have participated in the drafting and discussion of this memo: James E. Agenbroad, Harald Alvestrand, Andries Brouwer, Mark Davis, Martin J. DÈrst, Patrick FÈñltstrÈµm, Ned Freed, David Goldsmith, Tony Hansen, Edwin F. Hart, Paul Hoffman, David Hopwood, Kent Karlsson, Markus Kuhn, Michael Kung, Alain LaBontÈ¨, John Gardiner Myers, Dan Oscarsson, Murray Sargent, Markus Scherer, Keld Simonsen, Arnold Winkler, Kenneth Whistler and Misha Wolf.

Appendix B. Changes from RFC 2279

<63>
<64>
- Significantly shortened Introduction. No more mention of UTF-1 or UTF-7, of Transformation Formats.

<65>
- Straightened out terminology. UTF-8 now described in terms of an encoding form of the character number. UCS-2 and UCS-4 almost disappeared.

<66>
- Note warning against decoding of invalid sequences turned into a normative MUST NOT.

<67>
- New section about the BOM, mostly extracted and slightly adapted from RFC 2781.

<68>
- Updated a couple of references (10646-1:2000, Unicode 3.2, RFC 2978).

<69>
- Added TOC.

<70>
- Removed suggested UNICODE-1-1-UTF-8 MIME charset registration.

<71>
- New "Notational conventions" section about RFC 2119 and U+HHHH notation.

<72>
- Pointer to Unicode Consortium Policies added in "Versions of the standards" section.

<73>
- Added a fourth example with a non-BMP character and a BOM.

<74>
- Added a paragraph about U+2060 WORD JOINER.

<75>
- Enumerate more byte values impossible in UTF-8, either as a result of forbidding overlong sequences or of restricting to the UTF-16 accessible range.

<76>
- Added "IANA Considerations" section to ask that the UTF-8 entry in the charset registry point to this memo.

Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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