

Internet Draft  
<[draft-hoffman-utf16-04.txt](#)>  
June 1, 1999

Paul Hoffman  
Internet Mail Consortium  
Francois Yergeau  
Alis Technologies

## UTF-16, an encoding of ISO 10646

### 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/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>.

Copyright (C) The Internet Society (1999). All Rights Reserved.

### 1. Introduction

This document describes the UTF-16 encoding of Unicode/ISO-10646, addresses the issues of serializing UTF-16 as an octet stream for transmission over the Internet, defines MIME charset naming as described in [[CHARSET-REG](#)], and contains the registration for three MIME charset parameter values: UTF-16BE (big-endian), UTF-16LE (little-endian), and UTF-16.

#### 1.1 Background and motivation

The Unicode Standard [[UNICODE](#)] and ISO/IEC 10646 [[ISO-10646](#)] jointly define a coded character set (CCS), hereafter referred to as Unicode, which encompasses most of the world's writing systems [[WORKSHOP](#)]. UTF-16, the object of this specification, is one of the standard ways of encoding Unicode character data; it has the characteristics of encoding all currently defined characters (in plane 0, the BMP) in exactly two octets and of being able to encode all other characters likely to be defined (the next 16 planes) in exactly four octets.

The Unicode Standard further defines additional character properties

and other application details of great interest to implementors. 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, as well as not to assign characters outside of the 17 planes accessible to UTF-16.

The IETF policy on character sets and languages [[CHARPOLICY](#)] says that IETF protocols MUST be able to use the UTF-8 charset [[UTF-8](#)]. Although UTF-8 has many beneficial properties, such as the direct encoding of US-ASCII characters, re-synchronization after loss of octets and immunity to the byte-order issue (see 3.1 below), it is a variable-width encoding and is less dense than UTF-16 for characters whose values are between 0x0800 and 0xFFFF. Some products and network standards already specify UTF-16, making it an important encoding for the Internet.

## [1.2](#) Terminology

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 [RFC 2119](#) [[MUSTSHOULD](#)].

Throughout this document, character values are shown in hexadecimal notation. For example, "0x013C" is the character whose value is the character assigned the integer value 316 (decimal) in the CCS.

## [2](#). UTF-16 definition

In ISO 10646, each character is assigned a number, which Unicode calls the Unicode scalar value. This number is the same as the UCS-4 value of the character, and this document will refer to it as the "character value" for brevity. In the UTF-16 encoding, characters are represented using either one or two unsigned 16-bit integers, depending on the character value. Serialization of these integers for transmission as a byte stream is discussed in [Section 3](#).

The rules for how characters are encoded in UTF-16 are:

- Characters with values less than 0x10000 are represented as a single 16-bit integer with a value equal to that of the character number.
- Characters with values between 0x10000 and 0x10FFFF are represented by a 16-bit integer with a value between 0xD800 and 0xDBFF (within the so-called high-half zone or high surrogate area) followed by a 16-bit integer with a value between 0xDC00 and 0xDFFF (within the so-called low-half zone or low surrogate area).
- Characters with values greater than 0x10FFFF cannot be encoded in UTF-16.

Note: Values between 0xD800 and 0xDFFF are specifically reserved for use with UTF-16, and don't have any characters assigned to them.

## [2.1](#) Encoding UTF-16

Encoding of a single character from an ISO 10646 character value to UTF-16 proceeds as follows. Let  $U$  be the character number, no greater than 0x10FFFF.

- 1) If  $U < 0x10000$ , encode  $U$  as a 16-bit unsigned integer and terminate.
- 2) Let  $U' = U - 0x10000$ . Because  $U$  is less than or equal to 0x10FFFF,  $U'$  must be less than or equal to 0xFFFF. That is,  $U'$  can be represented in 20 bits.
- 3) Initialize two 16-bit unsigned integers,  $W1$  and  $W2$ , to 0xD800 and 0xDC00, respectively. These integers each have 10 bits free to encode the character value, for a total of 20 bits.
- 4) Assign the 10 high-order bits of the 20-bit  $U'$  to the 10 low-order bits of  $W1$  and the 10 low-order bits of  $U'$  to the 10 low-order bits of  $W2$ . Terminate.

Graphically, steps 2 through 4 look like:

$U' = \text{yyyyyyyyyyyyxxxxxxxxxx}$

$W1 = 110110\text{yyyyyyyyyy}$

$W2 = 110111\text{xxxxxxxxxx}$

## [2.2](#) Decoding UTF-16

Decoding of a single character from UTF-16 to an ISO 10646 character value proceeds as follows. Let  $W1$  be the next 16-bit integer in the sequence of integers representing the text. Let  $W2$  be the (eventual) next integer following  $W1$ .

- 1) If  $W1 < 0xD800$  or  $W1 > 0xDFFF$ , the character value  $U$  is the value of  $W1$ . Terminate.
- 2) Determine if  $W1$  is between 0xD800 and 0xDBFF. If not, the sequence is in error and no valid character can be obtained using  $W1$ . Terminate.
- 3) If there is no  $W2$  (that is, the sequence ends with  $W1$ ), or if  $W2$  is not between 0xDC00 and 0xDFFF, the sequence is in error. Terminate.
- 4) Construct a 20-bit unsigned integer  $U'$ , taking the 10 low-order bits of  $W1$  as its 10 high-order bits and the 10 low-order bits of  $W2$  as its [10](#) low-order bits.
- 5) Add 0x10000 to  $U'$  to obtain the character value  $U$ . Terminate.

Note that steps 2 and 3 indicate errors. Error recovery is not specified by this document. When terminating with an error in steps 2 and 3, it may be wise to set U to the value of W1 to help the caller diagnose the error and not lose information. Also note that a string decoding algorithm, as opposed to the single-character decoding described above, need not terminate upon detection of an error, if proper error reporting and/or recovery is provided.

### [3. Labelling UTF-16 text](#)

Appendix A of this specification contains registrations for three MIME charsets: "UTF-16BE", "UTF-16LE", and "UTF-16". MIME charsets represent the combination of a CCS (a coded character set) and a CES (a character encoding scheme). Here the CCS is Unicode/ISO 10646 and the CES is the same in all three cases, except for the serialization order of the octets in each character, and the external determination of which serialization is used.

This section describes which of the three labels to apply to a stream of text. [Section 4](#) describes how to interpret the labels on a stream of text.

#### [3.1 Definition of big-endian and little-endian](#)

Historically, computer hardware has processed two-octet entities such as 16-bit integers in one of two ways. So-called "big-endian" hardware handles two-octet entities with the higher-order octet first, that is at the lower address in memory; when written out to disk or to a network interface (serializing), the high-order octet thus appears first in the data stream. On the other hand, "Little-endian" hardware handles two-octet entities with the lower-order octet first. Hardware of both kinds is common today.

For example, the unsigned 16-bit integer that represents the decimal number 258 is 0x0102. The big-endian serialization of that number is the octet 0x01 followed by the octet 0x02. The little-endian serialization of that number is the octet 0x02 followed by the octet 0x01. The following C code fragment demonstrates a way to write 16-bit quantities to a file in big-endian order, irrespective of the hardware's native byte order.

```
void write_be(unsigned short u, FILE f) /* assume short is 16 bits */
{
    putc(u >> 8, f); /* output high-order byte */
    putc(u & 0xFF, f); /* then low-order */
}
```

The term "network byte order" has been used in many RFCs to indicate big-endian serialization, although that term has yet to be formally defined in a standards-track document. Although ISO 10646 prefers big-endian serialization (section 6.3 of [[ISO-10646](#)]), little-endian

order is also sometimes used on the Internet.

### [3.2](#) Byte order mark (BOM)

The Unicode Standard and ISO 10646 define the character "ZERO WIDTH NON-BREAKING SPACE" (0xFEFF), 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 NON-BREAKING SPACE" within text. This usage, suggested by Unicode [section 2.4](#) and ISO 10646 Annex F (informative), is to prepend a 0xFEFF character to a stream of Unicode characters as a "signature"; a receiver of such a serialized stream may then use the initial character both as a hint that the stream consists of Unicode characters and as a way to recognize the serialization order. In serialized UTF-16 prepended with such a signature, the order is big-endian if the first two octets are 0xFE followed by 0xFF; if they are 0xFF followed by 0xFE, the order is little-endian. Note that 0xFFFF is not a Unicode character, precisely to preserve the usefulness of 0xFEFF as a byte-order mark.

It is important to understand that the character 0xFEFF 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 0xFEFF 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 UTF-16 string into many parts, a part might begin with 0xFEFF because there was a zero-width non-breaking space at the beginning of that substring.

The Unicode standard further suggests that an initial 0xFEFF 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 NON-BREAKING 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.

In particular, in UTF-16 plain text it is likely, but not certain, that an initial 0xFEFF 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 NON-BREAKING SPACE" at the connection point. Also, some specifications mandate an initial 0xFEFF character in objects encoded in UTF-16 and specify that this signature is not part of the object.

### [3.3](#) Choosing a label for UTF-16 text

Any labelling application that uses UTF-16 character encoding, and

explicitly labels the text, and knows the serialization order of the characters in text, SHOULD label the text as either "UTF-16BE" or "UTF-16LE", whichever is appropriate based on the endianness of the text. This allows applications processing the text, but unable to look inside the text, to know the serialization definitively.

Text in the "UTF-16BE" charset MUST be serialized with the octets which make up a single 16-bit UTF-16 value in big-endian order. Systems labelling UTF-16BE text MUST NOT prepend a BOM to the text.

Text in the "UTF-16LE" charset MUST be serialized with the octets which make up a single 16-bit UTF-16 value in little-endian order. Systems labelling UTF-16LE text MUST NOT prepend a BOM to the text.

Any labelling application that uses UTF-16 character encoding, and puts an explicit charset label on the text, and does not know the serialization order of the characters in text, MUST label the text as "UTF-16", and SHOULD make sure the text starts with 0xFEFF.

An exception to the "SHOULD" rule of using "UTF-16BE" or "UTF-16LE" would occur with document formats that mandate a BOM in UTF-16 text, thereby requiring the use of the "UTF-16" tag only.

## [4. Interpreting text labels](#)

When a program sees text labelled as "UTF-16BE", "UTF-16LE", or "UTF-16", it can make some assumptions, based on the labelling rules given in the previous section. These assumptions allow the program to then process the text.

### [4.1 Interpreting text labelled as UTF-16BE](#)

Text labelled "UTF-16BE" can always be interpreted as being big-endian. The detection of an initial BOM does not affect de-serialization of text labelled as UTF-16BE. Finding 0xFF followed by 0xFE is an error since there is no Unicode character 0xFFFE.

### [4.2 Interpreting text labelled as UTF-16LE](#)

Text labelled "UTF-16LE" can always be interpreted as being little-endian. The detection of an initial BOM does not affect de-serialization of text labelled as UTF-16LE. Finding 0xFE followed by 0xFF is an error since there is no Unicode character 0xFFFE, which would be the interpretation of those octets under little-endian order.

### [4.3 Interpreting text labelled as UTF-16](#)

Text labelled with the "UTF-16" charset might be serialized in either big-endian or little-endian order. If the first two octets of the text is 0xFE followed by 0xFF, then the text can be interpreted as being big-endian. If the first two octets of the text is 0xFF followed by

0xFE, then the text can be interpreted as being little-endian. If the first two octets of the text is not 0xFE followed by 0xFF, and is not 0xFF followed by 0xFE, then the text SHOULD be interpreted as being big-endian.

All applications that process text with the "UTF-16" charset label MUST be able to read at least the first two octets of the text and be able to process those octets in order to determine the serialization order of the text. Applications that process text with the "UTF-16" charset label MUST NOT assume the serialization without first checking the first two octets to see if they are a big-endian BOM, a little-endian BOM, or not a BOM. All applications that process text with the "UTF-16" charset label MUST be able to interpret both big-endian and little-endian text.

## [5. Examples](#)

For the sake of example, let's suppose that there is a hieroglyphic character representing the Egyptian god Ra with character value 0x12345 (this character does not exist at present in Unicode).

The examples here all evaluate to the phrase:

\*=Ra

where the "\*" represents the Ra hieroglyph (0x12345).

Text labelled with UTF-16BE, without a BOM:

D8 08 DF 45 00 3D 00 52 00 61

Text labelled with UTF-16LE, without a BOM:

[08](#) D8 45 DF 3D 00 52 00 61 00

Big-endian text labelled with UTF-16, with a BOM:

FE FF D8 08 DF 45 00 3D 00 52 00 61

Little-endian text labelled with UTF-16, with a BOM:

FF FE 08 D8 45 DF 3D 00 52 00 61 00

## [6. 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, 2.0, and 2.1 as of this writing. Each new version 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 significant implementations and 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 consequences regarding MIME character encoding labels, to be discussed in [Appendix A](#).

## [7](#). Security considerations

UTF-16 is based on the ISO 10646 character set, which is frequently being added to, as described in [Section 6](#) and [Appendix A](#) of this document. Processors must be able to handle characters that are not defined at the time that the processor was created in such a way as to not allow an attacker to harm a recipient by including unknown characters.

Processors that handle any type of text, including text encoded as UTF-16, must be vigilant in checking for control characters that might reprogram a display terminal or keyboard. Similarly, processors that interpret text entities (such as looking for embedded programming code), must be careful not to execute the code without first alerting the recipient.

Text in UTF-16 may contain special characters, such as the OBJECT REPLACEMENT CHARACTER (0xFFFC), that might cause external processing, depending on the interpretation of the processing program and the availability of an external data stream that would be executed. This external processing may have side-effects that allow the sender of a message to attack the receiving system.

Implementors of UTF-16 need to consider the security aspects of how they handle illegal UTF-16 sequences (that is, sequences involving surrogate pairs that have illegal values or unpaired surrogates). It is conceivable that in some circumstances an attacker would be able to exploit an incautious UTF-16 parser by sending it an octet sequence that is not permitted by the UTF-16 syntax, causing it to behave in some anomalous fashion.

## [8](#). References

[CHARPOLICY] Alvestrand, H., "IETF Policy on Character Sets and Languages", [BCP 18](#), [RFC 2277](#), January 1998.

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

[HTTP-1.1] Fielding, R., et. al., "Hypertext Transfer Protocol --



HTTP/1.1", [RFC 2068](#), January 1997.

[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. Twelve amendments and two technical corrigenda have been published up to now. UTF-16 is described in Annex Q, published as Amendment 1. Many other amendments are currently at various stages of standardization.

[MUSTSHOULD] 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 2.0", ISBN 0-201-48345-9; with Unicode Technical Report #8, "The Unicode Standard, Version 2.1", <http://www.unicode.org/unicode/reports/tr8.html>.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[WORKSHOP] Weider, C., et. al., "Report of the IAB Character Set Workshop", [RFC 2130](#), April 1997.

## [9](#). Acknowledgments

Deborah Goldsmith wrote a great deal of the initial wording for this specification. Martin Duerst proposed numerous significant changes. Other significant contributors include:

Mati Allouche  
Walt Daniels  
Mark Davis  
Ned Freed  
Asmus Freytag  
Lloyd Honomichl  
Dan Kegel  
Murata Makoto  
Larry Masinter  
Markus Scherer  
Ken Whistler

Some of the text in this specification was copied from [[UTF-8](#)], and that document was worked on by many people. Please see the acknowledgments section in that document for more people who may have contributed indirectly to this document.

## [10](#). Changes between draft -03 and -04

2: Added note at the end of the section about 0xD800-0xDFFF being reserved for UTF-16.

3: Spelled out CCS and CES in the first paragraph. Also put a reference to [Appendix A](#) in the first paragraph. In the last paragraph, changed the last sentence to indicate that little-ending is already sometimes used on the Internet.

3.3: Changed the last paragraph to explain which kind of rules it applies to.

5: Changed "0x00012345" to "0x12345".

8: Changed the reference to [\[UNICODE\]](#).

## [A](#). Charset registrations

This memo is meant to serve as the basis for registration of three MIME charsets [\[CHARSET-REG\]](#). The proposed charsets are "UTF-16BE", "UTF-16LE", and "UTF-16". These strings label objects 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 and serialization schemes outlined above.

Note that "UTF-16BE", "UTF-16LE", and "UTF-16" are NOT suitable for use in media types under the "text" top-level type, because they do not encode line endings in the way required for MIME "text" media types. An exception to this is HTTP, which uses a MIME-like mechanism, but is exempt from the restrictions on the text top-level type (see section [19.4.1](#) of HTTP 1.1 [\[HTTP-1.1\]](#)).

It is noteworthy that the labels described here do not contain a version identification, referring generically to ISO/IEC 10646. This is intentional, the rationale being as follows:

A MIME charset 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.

The "Korean mess" (ISO/IEC 10646 amendment 5) is an incompatible change, in principle contradicting the appropriateness of a version independent MIME charset 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 charsets defined here will stay aligned with the previous version until and unless the IETF specifically decides otherwise.

#### [A.1](#) Registration for UTF-16BE

To: [ietf-charsets@iana.org](mailto:ietf-charsets@iana.org)  
Subject: Registration of new charset

Charset name(s): UTF-16BE

Published specification(s): This specification

Suitable for use in MIME content types under the  
"text" top-level type: No

Person & email address to contact for further information:  
Paul Hoffman <[phoffman@imc.org](mailto:phoffman@imc.org)>  
Francois Yergeau <[fyergerau@alis.com](mailto:fyergerau@alis.com)>

#### [A.2](#) Registration for UTF-16LE

To: [ietf-charsets@iana.org](mailto:ietf-charsets@iana.org)  
Subject: Registration of new charset

Charset name(s): UTF-16LE

Published specification(s): This specification

Suitable for use in MIME content types under the  
"text" top-level type: No

Person & email address to contact for further information:  
Paul Hoffman <[phoffman@imc.org](mailto:phoffman@imc.org)>  
Francois Yergeau <[fyergerau@alis.com](mailto:fyergerau@alis.com)>

#### [A.3](#) Registration for UTF-16

To: [ietf-charsets@iana.org](mailto:ietf-charsets@iana.org)  
Subject: Registration of new charset

Charset name(s): UTF-16

Published specification(s): This specification

Suitable for use in MIME content types under the  
"text" top-level type: No

Person & email address to contact for further information:

Paul Hoffman <phoffman@imc.org>

Francois Yergeau <fyergeau@alis.com>

[B](#). Authors' address

Paul Hoffman

Internet Mail Consortium

[127](#) Segre Place

Santa Cruz, CA 95060 USA

phoffman@imc.org

Francois Yergeau

Alis Technologies

100, boul. Alexis-Nihon, Suite 600

Montreal QC H4M 2P2 Canada

fyergeau@alis.com