Workgroup: Network Working Group
Internet-Draft: draft-yaoyang-dutf-01
Published: 25 March 2023
Intended Status: Experimental
Expires: 26 September 2023
Authors: Y. Yang

DUTF, a Dynamic Unicode Transformation Format

Abstract

The Unicode Standard and ISO/IEC 10646 jointly define a coded character set, referred to as Unicode, which encompasses most of the world's writing systems. Characters of the same language are arranged close to each other in the Unicode code table. This memo proposes a dynamic Unicode transformation format(DUTF). DUTF has the characteristic of preserving the full US-ASCII range, and uses XOR to calculate the offset value between the Unicode code point of adjacent non-ASCII characters in the source string, then encodes the result as a variable-length sequence of octets.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on 26 September 2023.

Copyright Notice

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 encoded individually and context-free. In most cases, a string will only contain one or two languages. Characters that belong to the same language are close to each other in the Unicode code table. Therefore, the character encoding can be effectively compressed by exploiting the correlation between adjacent characters.

DUTF, the object of this memo, has the capability to preserve the full US-ASCII [US-ASCII] range. For characters outside the US-ASCII range, DUTF calculates the offset value between adjacent characters in the source string using XOR, and then encodes the offset value as a variable-length sequence of octets. The number and value of octets depend on the Unicode code point of the current character and the

previous non-ASCII character in the source string. DUTF has the following characteristics (all values are hexadecimal):

*Characters in the range U+0000 to U+007F (US-ASCII repertoire) are represented as octets with values from 00 to 7F (7-bit US-ASCII values). As a result, a plain ASCII string is also a valid DUTF string.

*Characters other than ASCII are encoded as multiple octets.

*The highest bit of each octet determines whether the next octet belongs to the same character's encoding sequence. The remaining 7 bits hold the encoded value.

*The encoded value of the multi-octets represents the offset value between the Unicode code point of the current character and the previous non-ASCII character in the source string.

*Converting from DUTF to Unicode can be easily done.

*It is easy to find the starting point of each character boundary in a multi-octet stream.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Definition of DUTF

In DUTF, characters are encoded as sequences of 1 to n octets. For a single-octet sequence, the highest bit is set to 0 and the remaining 7 bits encode the character number. In sequences of n octets (n>1), the highest bit of the initial n-1 octets is set to 1, and the highest bit of the last octet is set to 0, with 7 bits available for encoding the offset value between the Unicode code point of the current character and the previous non-ASCII character in the source string.

Table 1 summarizes the format of these different variable-length octets. The letter x indicates bits available for encoding bits of the offset value.

Offset value range(hexadecimal)	DUTF octet sequence(binary)
0000 0000-0000 3FFF	1xxxxxxx 0xxxxxxx
0000 4000-001F FFFF	1xxxxxxx 1xxxxxxx 0xxxxxxx

Table 1

Encoding a character to DUTF proceeds as follows:

1. Determine whether the Unicode code point of the character is between 00000000 and 0000007F. If it is, the character belongs to the ASCII range and can be converted to an octet by simply converting the code point. Otherwise, continue to perform the following steps.

2. Use XOR operation to calculate the offset value between the Unicode code point of the current character and the previous non-ASCII character in the source string.

3. Determining the number of octets required based on the offset value and the conditions in the first column of Table 1. Prepare the highest bit of each octet as per the second column of Table 1.

4. Populate the x-marked bits with the binary representation of the offset value. Organize the binary representation of the offset value into groups of 7 bits, padding with zeros on the left if necessary. Then, starting from the rightmost group, use each group of 7 bits to replace the 7 x-marked bits of the corresponding octet in order, from left to right, until all x-marked bits have been filled in.

Decoding a DUTF character proceeds as follows:

1. Determine number of octets in the sequence, if it equals 1, then the current character belongs to the ASCII range, and the octet value is equal to the Unicode code point of the current character. Otherwise, continue to perform the following steps.

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

3. Distribute the 7 least significant bits from each octet of the sequence to the binary number. The first octet of the sequence corresponds to the 7 least significant bits of the binary number, the second octet corresponds to the next 7 least significant bits, and so on, until all bits have been assigned. The binary number is now equal to the offset value between the Unicode code point of the current character and the previous non-ASCII character in the source string.

4. XOR the offset value with the previous non-ASCII range character number to obtain the Unicode code point of the current character.

Implementations of the decoding algorithm above MUST protect against decoding invalid sequences. For instance, a naive implementation may

decode the invalid DUTF sequence 80 00 into the character U+0000. Decoding invalid sequences may have security consequences or cause other problems. See Security Considerations (Section 9) below.

3. Syntax of DUTF Byte Sequences

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

A DUTF string is a sequence of octets representing a sequence of Unicode characters. An octet sequence is valid DUTF only if it matches the following syntax, which is derived from the rules for encoding DUTF and is expressed in the ABNF of [RFC5234].

```
DUTF-octets = *( DUTF-char )
DUTF-char   = DUTF-1 / DUTF-2 / DUTF-3
DUTF-1      = %x00-7F
DUTF-2      = %x81-FF DUTF-tail
DUTF-3      = %x81-FF %x81-FF DUTF-tail
DUTF-tail   = %x00-7F
```

4. 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 [UNICODE-POLICIES]).

New versions, and in particular any incompatible changes, have consequences regarding MIME charset labels, to be discussed in MIME registration (Section 7).

5. 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. DUTF having a single-octet encoding unit, this last function is useless. BOM encoding is not fixed, only at the beginning of the stream, it will always be encoded as the octal sequence FF FD 03.

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

DUTF, 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.

*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.

6. Examples

The character sequence "A≠A."(U+0041 U+2262 U+0391 U+002E) is encoded in DUTF as Figure 1, which requires 6 octets, while using UTF-8 requires 7 octets.

```
--+-----+-----+--
41 E2 44 F3 43 2E
--+-----+-----+--
```

Figure 1

The character sequence "　　"(U+4E92 U+8054 U+7F51 U+5DE5 U+7A0B U+4EFB U+52A1 U+7EC4, meaning "The Internet Engineering Task Force" in Chinese) is encoded in DUTF as Figure 2, which requires 19 octets, while using UTF-8 requires 24 octets.

```
--------+--------+--------+-----+-----+-----+-----+-----
92 9D 01 C6 9D 03 85 FE 03 B4 45 EE 4F F0 69 DA 38 E5 58
--------+--------+--------+-----+-----+-----+-----+-----
```

Figure 2

The character sequence "　"(U+C0BC U+C131 U+C804 U+C790, meaning "Samsung Electronics" in Korean) is encoded in DUTF as Figure 3, which requires 9 octets, while using UTF-8 requires 12 octets.

```
--------+-----+-----+-----
BC 81 03 8D 03 B5 12 94 1F
--------+-----+-----+-----
```

Figure 3

The character sequence "　　"(U+3088 U+3053 U+306F U+307E U+3053 U+304F U+308A U+3064 U+3060 U+3044 U+304C U+304F, meaning "Yokohama National University" in Japanese) is encoded in DUTF as Figure 4, which requires 24 octets, while using UTF-8 requires 36 octets.

```
-----+-----+-----+-----+-----+-----
88 61 DB 01 BC 00 91 00 AD 00 9C 00
-----+-----+-----+-----+-----+-----
-----+-----+-----+-----+-----+-----
C5 01 EE 01 84 00 A4 00 88 00 83 00
-----+-----+-----+-----+-----+-----
```

Figure 4

The character sequence " "(U+233B4, a Chinese character meaning 'stump of tree'), prepended with a DUTF BOM, is encoded in DUTF as Figure 5, which requires 6 octets, while using UTF-8 requires 7 octets.

```
--------+--------
FF FD 03 CB 9A 0B
--------+--------
```

Figure 5

The character sequence "　hello　　"(U+4F60 U+597D U+0068 U+0065 U+006C U+006C U+006F U+3053 U+3093 U+306B U+3061 U+306F U+C548 U+B155 U+D558 U+C138 U+C694, a Chinese-English-Japanese-Korean mixed phrase, all of which mean "hello") is encoded in DUTF as Figure 6, which requires 34 octets, while using UTF-8 requires 41 octets.

```
--------+-----+--+--+--+--+--+--------+-----+-----
E0 9E 01 9D 2C 68 65 6C 6C 6F AE D2 01 C0 01 F8 01
--------+-----+--+--+--+--+--+--------+-----+-----
-----+-----+--------+--------+--------+-----+-----
8A 00 8E 00 A7 EA 03 9D E8 01 8D C8 01 E0 28 AC 0F
-----+-----+--------+--------+--------+-----+-----
```

Figure 6

The character sequence " ☀ "(U+5929 U+6C14 U+771F U+2600 U+FE0F U+1F44D, meaning "The weather is really nice") is encoded in DUTF as Figure 7, which requires 16 octets, while using UTF-8 requires 19 octets.

```
--------+-----+-----+--------+--------+--------
A9 B2 01 BD 6A 8B 36 9F A2 01 8F B0 03 C2 94 04
--------+-----+-----+--------+--------+--------
```

Figure 7

7. MIME Registration

This memo serves as the basis for registration of the MIME charset parameter for DUTF, according to [RFC2978]. The charset parameter value is "DUTF". 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. DUTF is suitable for use in MIME content types under the "text" top-level type.

It is noteworthy that the label "DUTF" 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 octets 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.

8. IANA Considerations

IANA is to register the charset found in Appendix A according to [RFC2978], using registration template found in this appendix.

9. Security Considerations

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

A particularly subtle form of this attack could be carried out against a parser which performs security-critical validity checks against the DUTF encoded form of its input, but interprets certain illegal octet sequences as characters. For example, a parser might prohibit the ACK character when encoded as the single-octet sequence 06, but allow the illegal two-octet sequence 86 00 and interpret it as a ACK character. Another example might be a parser which prohibits the octet sequence 2F 2E 2E 2F ("/../"), yet permits the illegal octet sequence AF 00 2E 2E 2F.

10. Acknowledgements

Some of the text in this specification was copied from [RFC3629] and [RFC2781].

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[ISO-10646] International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set (UCS)", ISO/IEC Standard 10646 2020, 2020, <https://www.iso.org/standard/76835.html>.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 15.0.0", ISBN 978-1-936213-32-0, 2022, <https://www.unicode.org/standard/versions/enumeratedversions.html#Unicode_15_0_0>.

11.2. Informative References

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, DOI 10.17487/RFC2781, February 2000, <https://www.rfc-editor.org/info/rfc2781>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, DOI 10.17487/RFC2978, October 2000, <https://www.rfc-editor.org/info/rfc2978>.

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

[UNICODE-POLICIES]
"Unicode Consortium Policies", <https://www.unicode.org/policies/index.html>.

Appendix A. Registration for DUTF

To: ietf-charsets@iana.org
Subject: Registration of new charset DUTF

Charset name: DUTF

Charset aliases: dutf

Suitability for use in MIME text: Body: ASCII compatible

Published specification(s): This specification

ISO 10646 equivalency table: This specification

Person & email address to contact for further information:
Yao Yang <yao.yang.sy@foxmail.com>

Intended usage: COMMON

Author's Address

Yao Yang
Room 501, Unit 4, Building 36, Hualong Yuan South
Changping District
Beijing, 102218
China

Phone: +86 182 0165 6971
Email: yao.yang.sy@foxmail.com