3.9 Unicode Encoding Forms

The Unicode Standard supports three character encoding forms: UTF-32, UTF-16, and UTF-8. Each encoding form maps the Unicode code points U+0000..U+D7FF and U+E000..U+10FFFF to unique code unit sequences. The size of the code unit is specified for each encoding form. This section presents the formal definition of each of these encoding forms.

D76 *Unicode scalar value:* Any Unicode code point except high-surrogate and low-surrogate code points.

- As a result of this definition, the set of Unicode scalar values consists of the ranges 0 to $D7FF_{16}$ and $E000_{16}$ to $10FFFF_{16}$, inclusive.

D77 *Code unit:* The minimal bit combination that can represent a unit of encoded text for processing or interchange.

- Code units are particular units of computer storage. Other character encoding standards typically use code units defined as 8-bit units—that is, *octets*. The Unicode Standard uses 8-bit code units in the UTF-8 encoding form, 16-bit code units in the UTF-16 encoding form, and 32-bit code units in the UTF-32 encoding form.
- A code unit is also referred to as a *code value* in the information industry.
- In the Unicode Standard, specific values of some code units cannot be used to represent an encoded character in isolation. This restriction applies to isolated surrogate code units in UTF-16 and to the bytes 80–FF in UTF-8. Similar restrictions apply for the implementations of other character encoding standards; for example, the bytes 81–9F, E0–FC in SJIS (Shift-JIS) cannot represent an encoded character by themselves.
- For information on use of `wchar_t` or other programming language types to represent Unicode code units, see "ANSI/ISO C wchar_t" in *Section 5.2, Programming Languages and Data Types*.

D78 *Code unit sequence:* An ordered sequence of one or more code units.

- When the code unit is an 8-bit unit, a code unit sequence may also be referred to as a *byte sequence*.
- A code unit sequence may consist of a single code unit.
- In the context of programming languages, the *value* of a *string* data type basically consists of a code unit sequence. Informally, a code unit sequence is itself just referred to as a *string*, and a *byte sequence* is referred to as a *byte string*. Care must be taken in making this terminological equivalence, however, because the formally defined concept of a string may have additional requirements or complications in programming languages. For example, a *string* is defined as a *pointer to char* in the C language and is conventionally terminated with a

NULL character. In object-oriented languages, a *string* is a complex object, with associated methods, and its value may or may not consist of merely a code unit sequence.

- Depending on the structure of a character encoding standard, it may be necessary to use a code unit sequence (of more than one unit) to represent a single encoded character. For example, the code unit in SJIS is a byte: encoded characters such as "a" can be represented with a single byte in SJIS, whereas ideographs require a sequence of two code units. The Unicode Standard also makes use of code unit sequences whose length is greater than one code unit.

D79 A *Unicode encoding form* assigns each Unicode scalar value to a unique code unit sequence.

- For historical reasons, the Unicode encoding forms are also referred to as *Unicode* (or *UCS*) *transformation formats* (UTF). That term is actually ambiguous between its usage for encoding forms and encoding schemes.
- The mapping of the set of Unicode scalar values to the set of code unit sequences for a Unicode encoding form is *one-to-one*. This property guarantees that a reverse mapping can always be derived. Given the mapping of any Unicode scalar value to a particular code unit sequence for a given encoding form, one can derive the original Unicode scalar value unambiguously from that code unit sequence.
- The mapping of the set of Unicode scalar values to the set of code unit sequences for a Unicode encoding form is not *onto*. In other words, for any given encoding form, there exist code unit sequences that have no associated Unicode scalar value.
- To ensure that the mapping for a Unicode encoding form is one-to-one, *all* Unicode scalar values, including those corresponding to noncharacter code points and unassigned code points, must be mapped to unique code unit sequences. Note that this requirement does not extend to high-surrogate and low-surrogate code points, which are excluded by definition from the set of Unicode scalar values.

D80 *Unicode string:* A code unit sequence containing code units of a particular Unicode encoding form.

- In the rawest form, Unicode strings may be implemented simply as arrays of the appropriate integral data type, consisting of a sequence of code units lined up one immediately after the other.
- A single Unicode string must contain only code units from a single Unicode encoding form. It is not permissible to mix forms within a string.

D81 *Unicode 8-bit string:* A Unicode string containing only UTF-8 code units.

D82 *Unicode 16-bit string:* A Unicode string containing only UTF-16 code units.

D83 *Unicode 32-bit string:* A Unicode string containing only UTF-32 code units.

D84 *Ill-formed:* A Unicode code unit sequence that purports to be in a Unicode encoding form is called *ill-formed* if and only if it does *not* follow the specification of that Unicode encoding form.

- Any code unit sequence that would correspond to a code point outside the defined range of Unicode scalar values would, for example, be ill-formed.
- UTF-8 has some strong constraints on the possible byte ranges for leading and trailing bytes. A violation of those constraints would produce a code unit sequence that could not be mapped to a Unicode scalar value, resulting in an ill-formed code unit sequence.

D84a *Ill-formed code unit subsequence:* A non-empty subsequence of a Unicode code unit sequence X which does not contain any code units which also belong to any minimal well-formed subsequence of X.

- In other words, an ill-formed code unit subsequence cannot overlap with a minimal well-formed subsequence.

D85 *Well-formed:* A Unicode code unit sequence that purports to be in a Unicode encoding form is called *well-formed* if and only if it *does* follow the specification of that Unicode encoding form.

D85a *Minimal well-formed code unit subsequence:* A well-formed Unicode code unit sequence that maps to a single Unicode scalar value.

- For UTF-8, see the specification in D92 and *Table 3-7*.
- For UTF-16, see the specification in D91.
- For UTF-32, see the specification in D90.

A well-formed Unicode code unit sequence can be partitioned into one or more minimal well-formed code unit sequences for the given Unicode encoding form. Any Unicode code unit sequence can be partitioned into subsequences that are either well-formed or ill-formed. The sequence as a whole is well-formed if and only if it contains no ill-formed subsequence. The sequence as a whole is ill-formed if and only if it contains at least one ill-formed subsequence.

D86 *Well-formed UTF-8 code unit sequence:* A well-formed Unicode code unit sequence of UTF-8 code units.

- The UTF-8 code unit sequence <41 C3 B1 42> is well-formed, because it can be partitioned into subsequences, all of which match the specification for UTF-8 in *Table 3-7*. It consists of the following minimal well-formed code unit subsequences: <41>, <C3 B1>, and <42>.
- The UTF-8 code unit sequence <41 C2 C3 B1 42> is ill-formed, because it contains one ill-formed subsequence. There is no subsequence for the C2 byte which matches the specification for UTF-8 in *Table 3-7*. The code unit

sequence is partitioned into one minimal well-formed code unit subsequence, <41>, followed by one ill-formed code unit subsequence, <C2>, followed by two minimal well-formed code unit subsequences, <C3 B1> and <42>.

- In isolation, the UTF-8 code unit sequence <C2 C3> would be ill-formed, but in the context of the UTF-8 code unit sequence <41 C2 C3 B1 42>, <C2 C3> does not constitute an ill-formed code unit subsequence, because the C3 byte is actually the first byte of the minimal well-formed UTF-8 code unit subsequence <C3 B1>. Ill-formed code unit subsequences do not overlap with minimal well-formed code unit subsequences.

D87 *Well-formed UTF-16 code unit sequence:* A well-formed Unicode code unit sequence of UTF-16 code units.

D88 *Well-formed UTF-32 code unit sequence:* A well-formed Unicode code unit sequence of UTF-32 code units.

D89 *In a Unicode encoding form:* A Unicode string is said to be *in* a particular Unicode encoding form if and only if it consists of a well-formed Unicode code unit sequence of that Unicode encoding form.

- A Unicode string consisting of a well-formed UTF-8 code unit sequence is said to be *in UTF-8*. Such a Unicode string is referred to as a *valid UTF-8 string*, or a *UTF-8 string* for short.
- A Unicode string consisting of a well-formed UTF-16 code unit sequence is said to be *in UTF-16*. Such a Unicode string is referred to as a *valid UTF-16 string*, or a *UTF-16 string* for short.
- A Unicode string consisting of a well-formed UTF-32 code unit sequence is said to be *in UTF-32*. Such a Unicode string is referred to as a *valid UTF-32 string*, or a *UTF-32 string* for short.

Unicode strings need not contain well-formed code unit sequences under all conditions. This is equivalent to saying that a particular Unicode string need not be *in* a Unicode encoding form.

- For example, it is perfectly reasonable to talk about an operation that takes the two Unicode 16-bit strings, <004D D800> and <DF02 004D>, each of which contains an ill-formed UTF-16 code unit sequence, and concatenates them to form another Unicode string <004D D800 DF02 004D>, which contains a well-formed UTF-16 code unit sequence. The first two Unicode strings are not *in* UTF-16, but the resultant Unicode string is.
- As another example, the code unit sequence <C0 80 61 F3> is a Unicode 8-bit string, but does not consist of a well-formed UTF-8 code unit sequence. That code unit sequence could not result from the specification of the UTF-8 encoding form and is thus ill-formed. (The same code unit sequence could, of course, be well-formed in the context of some other character encoding standard using 8-bit code units, such as ISO/IEC 8859-1, or vendor code pages.)

If a Unicode string *purports* to be *in* a Unicode encoding form, then it must not contain any ill-formed code unit subsequence.

If a process which verifies that a Unicode string is in a Unicode encoding form encounters an ill-formed code unit subsequence in that string, then it must not identify that string as being in that Unicode encoding form.

A process which interprets a Unicode string must not interpret any ill-formed code unit subsequences in the string as characters. (See conformance clause C10.) Furthermore, such a process must not treat any adjacent well-formed code unit sequences as being part of those ill-formed code unit sequences.

Table 3-4 gives examples that summarize the three Unicode encoding forms.

Table 3-4. Examples of Unicode Encoding Forms

Code Point	Encoding Form	Code Unit Sequence
U+004D	UTF-32	0000004D
	UTF-16	004D
	UTF-8	4D
U+0430	UTF-32	00000430
	UTF-16	0430
	UTF-8	D0 B0
U+4E8C	UTF-32	00004E8C
	UTF-16	4E8C
	UTF-8	E4 BA 8C
U+10302	UTF-32	00010302
	UTF-16	D800 DF02
	UTF-8	F0 90 8C 82

UTF-32

D90 *UTF-32 encoding form:* The Unicode encoding form that assigns each Unicode scalar value to a single unsigned 32-bit code unit with the same numeric value as the Unicode scalar value.

- In UTF-32, the code point sequence <004D, 0430, 4E8C, 10302> is represented as <0000004D 00000430 00004E8C 00010302>.
- Because surrogate code points are not included in the set of Unicode scalar values, UTF-32 code units in the range $0000D800_{16}$..$0000DFFF_{16}$ are ill-formed.
- Any UTF-32 code unit greater than $0010FFFF_{16}$ is ill-formed.

For a discussion of the relationship between UTF-32 and UCS-4 encoding form defined in ISO/IEC 10646, see *Appendix C.2, Encoding Forms in ISO/IEC 10646*.

UTF-16

D91 *UTF-16 encoding form:* The Unicode encoding form that assigns each Unicode scalar value in the ranges U+0000..U+D7FF and U+E000..U+FFFF to a single unsigned 16-bit code unit with the same numeric value as the Unicode scalar value, and that assigns each Unicode scalar value in the range U+10000..U+10FFFF to a surrogate pair, according to *Table 3-5*.

- In UTF-16, the code point sequence <004D, 0430, 4E8C, 10302> is represented as <004D 0430 4E8C D800 DF02>, where <D800 DF02> corresponds to U+10302.
- Because surrogate code points are not Unicode scalar values, isolated UTF-16 code units in the range $D800_{16}$..$DFFF_{16}$ are ill-formed.

Table 3-5 specifies the bit distribution for the UTF-16 encoding form. Note that for Unicode scalar values equal to or greater than U+10000, UTF-16 uses surrogate pairs. Calculation of the surrogate pair values involves subtraction of 10000_{16}, to account for the starting offset to the scalar value. ISO/IEC 10646 specifies an equivalent UTF-16 encoding form. For details, see *Appendix C.3, UTF-8 and UTF-16*.

Table 3-5. UTF-16 Bit Distribution

Scalar Value	UTF-16
xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx
000uuuuuxxxxxxxxxxxxxxxx	110110wwwwxxxxxx 110111xxxxxxxxxx

Note: wwww = uuuuu - 1

UTF-8

D92 *UTF-8 encoding form:* The Unicode encoding form that assigns each Unicode scalar value to an unsigned byte sequence of one to four bytes in length, as specified in *Table 3-6* and *Table 3-7*.

- In UTF-8, the code point sequence <004D, 0430, 4E8C, 10302> is represented as <4D D0 B0 E4 BA 8C F0 90 8C 82>, where <4D> corresponds to U+004D, <D0 B0> corresponds to U+0430, <E4 BA 8C> corresponds to U+4E8C, and <F0 90 8C 82> corresponds to U+10302.
- Any UTF-8 byte sequence that does not match the patterns listed in *Table 3-7* is ill-formed.
- Before the Unicode Standard, Version 3.1, the problematic “non-shortest form” byte sequences in UTF-8 were those where BMP characters could be represented in more than one way. These sequences are ill-formed, because they are not allowed by *Table 3-7*.

- Because surrogate code points are not Unicode scalar values, any UTF-8 byte sequence that would otherwise map to code points U+D800..U+DFFF is ill-formed.

Table 3-6 specifies the bit distribution for the UTF-8 encoding form, showing the ranges of Unicode scalar values corresponding to one-, two-, three-, and four-byte sequences. For a discussion of the difference in the formulation of UTF-8 in ISO/IEC 10646, see *Appendix C.3, UTF-8 and UTF-16*.

Table 3-6. UTF-8 Bit Distribution

Scalar Value	First Byte	Second Byte	Third Byte	Fourth Byte
00000000 0xxxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
000uuuuu zzzzyyyy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

Table 3-7 lists all of the byte sequences that are well-formed in UTF-8. A range of byte values such as A0..BF indicates that any byte from A0 to BF (inclusive) is well-formed in that position. Any byte value outside of the ranges listed is ill-formed. For example:

- The byte sequence <C0 AF> is *ill-formed*, because C0 is not well-formed in the "First Byte" column.
- The byte sequence <E0 9F 80> is *ill-formed*, because in the row where E0 is well-formed as a first byte, 9F is not well-formed as a second byte.
- The byte sequence <F4 80 83 92> is *well-formed*, because every byte in that sequence matches a byte range in a row of the table (the last row).

Table 3-7. Well-Formed UTF-8 Byte Sequences

Code Points	First Byte	Second Byte	Third Byte	Fourth Byte
U+0000..U+007F	00..7F			
U+0080..U+07FF	C2..DF	80..BF		
U+0800..U+0FFF	E0	***A0***..BF	80..BF	
U+1000..U+CFFF	E1..EC	80..BF	80..BF	
U+D000..U+D7FF	ED	80..***9F***	80..BF	
U+E000..U+FFFF	EE..EF	80..BF	80..BF	
U+10000..U+3FFFF	F0	***90***..BF	80..BF	80..BF
U+40000..U+FFFFF	F1..F3	80..BF	80..BF	80..BF
U+100000..U+10FFFF	F4	80..***8F***	80..BF	80..BF

In *Table 3-7*, cases where a trailing byte range is not 80..BF are shown in bold italic to draw attention to them. These exceptions to the general pattern occur only in the second byte of a sequence.

As a consequence of the well-formedness conditions specified in *Table 3-7*, the following byte values are disallowed in UTF-8: C0–C1, F5–FF.

Encoding Form Conversion

D93 *Encoding form conversion:* A conversion defined directly between the code unit sequences of one Unicode encoding form and the code unit sequences of another Unicode encoding form.

- In implementations of the Unicode Standard, a typical API will logically convert the input code unit sequence into Unicode scalar values (code points) and then convert those Unicode scalar values into the output code unit sequence. Proper analysis of the encoding forms makes it possible to convert the code units directly, thereby obtaining the same results but with a more efficient process.
- A conformant encoding form conversion will treat any ill-formed code unit sequence as an error condition. (See conformance clause C10.) This guarantees that it will neither interpret nor emit an ill-formed code unit sequence. Any implementation of encoding form conversion must take this requirement into account, because an encoding form conversion implicitly involves a verification that the Unicode strings being converted do, in fact, contain well-formed code unit sequences.

Constraints on Conversion Processes

The requirement not to interpret any ill-formed code unit subsequences in a string as characters (see conformance clause C10) has important consequences for conversion processes. Such processes may, for example, interpret UTF-8 code unit sequences as Unicode character sequences. If the converter encounters an ill-formed UTF-8 code unit sequence which starts with a valid first byte, but which does not continue with valid successor bytes (see *Table 3-7*), it *must not* consume the successor bytes as part of the ill-formed subsequence whenever those successor bytes themselves constitute part of a well-formed UTF-8 code unit subsequence.

If an implementation of a UTF-8 conversion process stops at the first error encountered, without reporting the end of any ill-formed UTF-8 code unit subsequence, then the requirement makes little practical difference. However, the requirement does introduce a significant constraint if the UTF-8 converter continues past the point of a detected error, perhaps by substituting one or more U+FFFD replacement characters for the uninterpretable, ill-formed UTF-8 code unit subsequence. For example, with the input UTF-8 code unit sequence <C2 41 42>, such a UTF-8 conversion process must not return <U+FFFD> or <U+FFFD, U+0042>, because either of those outputs would be the result of misinterpreting a well-formed subsequence as being part of the ill-formed subsequence. The expected return value for such a process would instead be <U+FFFD, U+0041, U+0042>.

For a UTF-8 conversion process to consume valid successor bytes is not only non-conformant, but also leaves the converter open to security exploits. See Unicode Technical Report #36, "Unicode Security Considerations."

Although a UTF-8 conversion process is required to never consume well-formed subsequences as part of its error handling for ill-formed subsequences, such a process is not otherwise constrained in how it deals with any ill-formed subsequence itself. An ill-formed subsequence consisting of more than one code unit could be treated as a single error or as multiple errors.

For example, in processing the UTF-8 code unit sequence <F0 80 80 41>, the only formal requirement mandated by Unicode conformance for a converter is that the <41> be processed and correctly interpreted as <U+0041>. The converter could return <U+FFFD, U+0041>, handling <F0 80 80> as a single error, or <U+FFFD, U+FFFD, U+FFFD, U+0041>, handling each byte of <F0 80 80> as a separate error, or could take other approaches to signalling <F0 80 80> as an ill-formed code unit subsequence.

U+FFFD Substitution of Maximal Subparts

An increasing number of implementations are adopting the handling of ill-formed subsequences as specified in the W3C standard for encoding to achieve consistent U+FFFD replacements. See:

http://www.w3.org/TR/encoding/

Although the Unicode Standard does not require this practice for conformance, the following text describes this practice and gives detailed examples.

D93a *Unconvertible offset:* An offset in a code unit sequence for which no code unit subsequence starting at that offset is well-formed.

D93b *Maximal subpart of an ill-formed subsequence:* The longest code unit subsequence starting at an unconvertible offset that is either:

a. the initial subsequence of a well-formed code unit sequence, or

b. a subsequence of length one.

This definition of the maximal subpart is used in describing how far to advance processing when making substitutions: always process at least one code unit, or as many code units as match the beginning of a well-formed character, up to the point where the next code unit would make it ill-formed, that is, an offset is reached that does not continue this partial character.

Or stated more formally:

> ***Whenever an unconvertible offset is reached during conversion of a code unit sequence:***
>
> ***1. The maximal subpart at that offset is replaced by a single U+FFFD.***

2. The conversion proceeds at the offset immediately after the maximal subpart.

This practice of substituting maximal subparts can be trivially applied to the UTF-32 or UTF-16 encoding forms, but is primarily of interest when converting UTF-8 strings.

Unless the beginning of an ill-formed subsequence matches the beginning of some well-formed sequence, this practice replaces almost every byte of an ill-formed UTF-8 sequence with one U+FFFD. For example, every byte of a "non-shortest form" sequence (see Definition D92), or of a truncated version thereof, is replaced, as shown in *Table 3-8*. (The interpretation of "non-shortest form" sequences has been forbidden since the publication of Corrigendum #1.)

Table 3-8. U+FFFD for Non-Shortest Form Sequences

Bytes	C0	AF	E0	80	BF	F0	81	82	41
Output	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	0041

Also, every byte of a sequence that would correspond to a surrogate code point, or of a truncated version thereof, is replaced with one U+FFFD, as shown in *Table 3-9*. (The interpretation of such byte sequences has been forbidden since Unicode 3.2.)

Table 3-9. U+FFFD for Ill-Formed Sequences for Surrogates

Bytes	ED	A0	80	ED	BF	BF	ED	AF	41
Output	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	0041

Finally, every byte of a sequence that would correspond to a code point beyond U+10FFFF, and any other byte that does not contribute to a valid sequence, is also replaced with one U+FFFD, as shown in *Table 3-10*.

Table 3-10. U+FFFD for Other Ill-Formed Sequences

Bytes	F4	91	92	93	FF	41	80	BF	42
Output	FFFD	FFFD	FFFD	FFFD	FFFD	0041	FFFD	FFFD	0042

Only when a sequence of two or three bytes is a truncated version of a sequence which is otherwise well-formed to that point, is more than one byte replaced with a single U+FFFD, as shown in *Table 3-11*.

Table 3-11. U+FFFD for Truncated Sequences

Bytes	E1	80	E2	F0	91	92	F1	BF	41
Output	FFFD		FFFD	FFFD			FFFD		0041

For a discussion of the generalization of this approach for conversion of other character sets to Unicode, see *Section 5.22, U+FFFD Substitution in Conversion*.