UTC /1999-011

# Title: Summary of Normative Changes to Conformance for Unicode 3.0

Date: May 21, 1999
Source: Editorial Committee
Action: For Approval at June, 1999 UTC meeting

The following clauses, snipped from the working draft for the book, The Unicode Standard, Version 3.0, constitute the normative clauses in Chapter 3, Conformance, which have been significantly altered or which have been added to the standard since the publication of Version 2.0. (The Bidi section is omitted, since it has been covered on separate agenda topics, with other committee documents.)

The main new material has to do with transformation formats, and has already been agreed to in principle by UTC. However, for completeness sake, all new and modified clauses are repeated below, with changes highlighted for those instances with just minor differences. This material is not presented to reopen everything for discussion and debate, but rather to give the UTC one last look before we go to press, so that we can get a definitive endorsement of the normative content in the standard regarding conformance.

D9a *deprecated character:* a character whose use is strongly discouraged. Such characters are retained in the standard, but should not be used.

- Deprecated characters are retained in the standard so that previously conforming data stays conformant in future versions of the standard.
- Obsolete characters are historical. They do not occur in modern text, but they do not constitute deprecated characters.

D10a *case property:* a property of characters in certain alphabets whereby certain characters are considered variants of a single letter. (See *Section 4.1, Case—Normative.*)

D10b *numeric value property:* a property of characters used to represent numbers. (See *Section 4.6, Numeric Value—Normative.*)

### *Compatibility Decomposition*

D20 *compatibility decomposition:* the decomposition of a character which results from recursively applying *both* the compatibility mappings *and* the canonical mappings found in the names list of *Section 14.1, Character Names List* and those described in *Section 3.11, Conjoining Jamo Behavior* until no characters can be further decomposed, and then reordering non-spacing marks according to *Section 3.10, Canonical Ordering Behavior.*

### *Transformations*

*C11* *When a process interprets a byte sequence in a Unicode Transformation Format, it shall interpret that byte sequence in accordance with the character semantics established by this standard for the corresponding Unicode character sequence.*

*C12* *When a process generates data in a Unicode Transformation Format, it shall not emit ill-formed byte sequences. When a process interprets data in a Unicode Transformation format, it shall treat illegal byte sequences as an error condition.*

## 3.8 Transformations

There is more than one representation of Unicode data that can be conformant to the Unicode Standard. Chief among them is UTF-8, discussed in *Section 2.3, Encoding Forms* and *Appendix C.3, Transformation Formats.* In addition, there are compression transformations such as the one described in the Unicode Technical Report #6, "A Standard Compression Scheme for Unicode" on the CD-ROM or the up-to-date version on the Unicode web site.

*D29* A *Unicode transformation format* (UTF) is a mapping from each Unicode coded character sequence to a unique sequence of code values.

- These code values are particular units of computer storage specified by the transformation format, typically bytes. In that case, a code value sequence can be referred to as a *byte sequence.*

Since every Unicode coded character sequence maps to a unique sequence of code values in a given UTF, a reverse mapping can be derived. Thus every UTF supports *lossless round-trip transcoding:* mapping from any Unicode coded character sequence S to a sequence of code values and back will produce S again. To ensure round-trip transcoding, a UTF mapping *must also* map the 16-bit codes that are not valid Unicode scalar values to unique code value sequences. These invalid 16-bit codes are $FFFE_{16}$, $FFFF_{16}$, and unpaired surrogates.

*D30* For a given UTF, a code value sequence that cannot be produced from any sequence of 16-byte values is called an *ill-formed code value sequence.*

*D31* For a given UTF, a code value sequence that cannot be mapped back to a sequence of 16-byte values is called an *illegal code value sequence.*

- For example, in UTF-8 every code value of the form $110xxxxx_2$ *must* be followed with a code value of the form $10xxxxxx_2$. A sequence such as $110xxxxx_2$ $0xxxxxx_2$ is illegal, and must never be generated. When faced with this illegal code value sequence while transforming or interpreting, a UTF-8 conformant process must treat the first code value $110xxxxx_2$ as an illegal termination error: for example, either signaling an error, filtering the code value out, or representing the code value with a marker such as U+FFFD REPLACEMENT CHARACTER. In the latter two cases, it will continue processing at the second code value $0xxxxxx_2$.

*D32* For a given UTF, an ill-formed code value sequence that is not illegal is called an *irregular code value sequence.*

- To make implementations simpler and faster, some transformation formats may allow irregular code value sequences without requiring error handling. For example, UTF-8 allows non-shortest code value sequences to be interpreted: a UTF-8 conformant process may map the code value sequence C0 80 ($11000000_2$ $10000000_2$) to the Unicode value U+0000, even though a UTF-8 conformant process shall *never* generate that code value sequence—it shall generate the sequence 00 ($00000000_2$) instead. A conformant process shall not use irregular code value sequences to encode out-of-band information.

*D33* UTF-16BE is the Unicode Transformation Format that serializes a Unicode value as a sequence of two bytes, in Big Endian format. An initial sequence corresponding to U+FEFF is interpreted as a ZERO WIDTH NO-BREAK SPACE.

- In UTF-16BE, <004D 0061 0072 006B> is serialized as <00 4D 00 61 00 72 00 6B>

*D34* UTF-16LE is the Unicode Transformation Format that serializes a Unicode value as a sequence of two bytes, in Little Endian format. An initial sequence corresponding to U+FEFF is interpreted as a ZERO WIDTH NO-BREAK SPACE.

- In UTF-16LE, <004D 0061 0072 006B> is serialized as <4D 00 61 00 72 00 6B 00>

*D35* UTF-16 is the Unicode Transformation Format that serializes a Unicode value as a sequence of two bytes, in either Big Endian or Little Endian format. An initial sequence corresponding to U+FEFF is interpreted as a *byte order mark*, and is used to distinguish between the two endians for the rest of the text. It is not considered part of the content of the text. A serialization of Unicode values into UTF-16 may or may not begin with a byte order mark.

- In UTF-16, <004D 0061 0072 006B> is serialized as <FF FE 4D 00 61 00 72 00 6B 00> or <FE FF 00 4D 00 61 00 72 00 6B> or <00 4D 00 61 00 72 00 6B>

*D36* UTF-8 is the Unicode Transformation Format that serializes a Unicode scalar value as a sequence of one to four bytes, as specified in *Table 3-1*.

- In UTF-8, <004D 0061 0072 006B> is serialized as <4D 61 72 6B>

*Table 3-1* specifies the bit distribution from a Unicode character (or surrogate pair) into the one- to four-byte values of the corresponding UTF-8 sequence. Note that the four-byte form for surrogate pairs involves an addition of $10000_{16}$ to account for the starting offset to the encoded values referenced by surrogates. The definition of UTF-8 in Amendment 2 to ISO/IEC 10646 can also use five- and six-byte sequences to encode characters that are outside the range of the Unicode character set; those five- and six-byte sequences are undefined for the use of UTF-8 as a transformation of Unicode characters.

## Table 3-1. UTF-8 Bit Distribution

| Scalar Value | UTF-16 | 1st Byte | 2nd Byte | 3rd Byte | 4th Byte |
|---|---|---|---|---|---|
| 000000000xxxxxxx | 000000000xxxxxxx | 0xxxxxxx | | | |
| 00000yyyyyxxxxxx | 00000yyyyyxxxxxx | 110yyyyy | 10xxxxxx | | |
| zzzzyyyyyyxxxxxx | zzzzyyyyyyxxxxxx | 1110zzzz | 10yyyyyy | 10xxxxxx | |
| uuuuuzzzzyyyyyyxxxxxx | 110110wwwwzzzzyy+ 110111yyyyxxxxxx | 11110uuu[a] | 10uuzzzz | 10yyyyyy | 10xxxxxx |

a. where uuuuu = wwww + 1 (to account for addition of $10000_{16}$ as in *Section 3.7, Surrogates*)

When converting a Unicode scalar value to UTF-8, the shortest form that can represent those values shall be used. This preserves uniqueness of encoding. For example, the Unicode binary value <0000000000000001> is encoded as <00000001>, not as <11000000 10000001>. The latter is an example of an irregular UTF-8 byte sequence. *Irregular UTF-8 sequences shall not be used for encoding any other information.*

When converting from UTF-8 to a Unicode scalar value, implementations do not need to check that the shortest encoding is being used. This simplifies the conversion algorithm.