

4.0 Additional Character Semantics

The following section describes additional character semantics that more precisely specify the use of characters already present in volumes 1 and 2.

4.1 Fraction Slash

The fraction slash is used to form numeric fractions. The standard form of a fraction built using the fraction slash is defined as follows: any sequence of one or more decimal digits, followed by the fraction slash, followed by any sequence of one or more decimal digits. Such a fraction should be displayed as unit, such as $\frac{6}{8}$ or as $\frac{6}{8}$. The precise choice of display can depend upon additional formatting information.

If the displaying software is incapable of mapping the fraction to a unit, then it can also be displayed as a simple linear sequence as a fall-back: e.g. 6/8. If the fraction is to be separated from a previous number, then a space can be used, choosing the appropriate width (normal, thin, zero-width, etc.). For example, 1 + *zero-width space* + 6 + *fraction slash* + 8 displays as 1⁶/₈.

4.2 Dotful I:

A *dotted* (normal) *i* followed by a top non-spacing mark loses the dot in rendering. Thus in the word *naïve* the *ï* could be spelled with *i* + *diaeresis*. Just as Cyrillic A is not equivalent to Latin A, a *dotted-i* is not equivalent to a Turkish *dotless-i* + *overdot* (*i* + \cdot), nor are other cases of accents equivalent (*i* + $\grave{\cdot}$ \neq *i* + $\ddot{\cdot}$).

To express the forms sometimes used in the Baltic (where the dot is retained under a top accent), use *i* + *overdot* + *accent*:

i + $\dot{\circ}$ + $\acute{\circ}$ \Rightarrow $\acute{\dot{i}}$

i + $\acute{\circ}$ + $\dot{\circ}$ \Rightarrow $\dot{\acute{i}}$

4.3 Surrounding Non-Spacing Marks

Surrounding non-spacing marks surround all previous characters up to and including the base character. They thus successively surround previous surrounding non-spacing marks. For example:

a + \square + $\ddot{\circ}$ + \circ \Rightarrow $\textcircled{\square\ddot{a}}$

4.4 Unicode Character Equivalence

Unicode 1.0 defines two sequences of non-spacing marks to be equivalent if they do not interact typographically (Volume 1, p. 18, ¶ 2 & p. 21, item 3). This works well in cases such as the examples cited, such as NON-SPACING DOT BELOW and NON-SPACING DOT ABOVE, which clearly do not interact.

However, a general, consistent method of determining character string equivalence requires an explicit algorithm, which is provided below. Basically it works as follows. Every Unicode non-spacing mark has an associated non-spacing priority (spacing marks have a null priority). Whenever a character is encountered that has a non-null priority, a reordering algorithm is

invoked. Essentially, any sequence of non-null priority marks is sorted based on the priority. This algorithm represents a logical description of the process: optimized algorithms can be used in implementations as long as they are equivalent (i.e., they produce the same result).

1. Two sequences of characters are equivalent if their canonical ordering representation is identical.
2. The canonical ordering representation of a string of characters is determined in the following way:
 - a. Decompose all precomposed characters in the string, based upon Appendix I: Unicode 1.1 Character List, p. 43. This will form the maximal decomposition of the string into component characters.
 - b. Assign a canonical priority to each character, based upon Appendix D: Canonical Ordering Priorities, p. 23. Let $p(A)$ be the priority of the character A .
 - c. Sort the string by successively exchanging each pair (A, B) of adjacent characters wherever $p(B) \neq 0$ & $p(A) > p(B)$.

Examples:

$a + \textit{underdot} + \textit{diaeresis}$ \rightarrow $a + \textit{diaeresis} + \textit{underdot}$
 $a + \textit{diaeresis} + \textit{underdot}$ \rightarrow $a + \textit{diaeresis} + \textit{underdot}$


Since *underdot* has a larger non-spacing priority than *circumflex*, the algorithm will return the *a*, then the *diaeresis*, then the *underdot*. However, since *diaeresis* and *breve* have the same non-spacing priority (because they interact typographically), they do not rearrange.

$a + \textit{breve} + \textit{diaeresis}$  $a + \textit{diaeresis} + \textit{breve}$
 $a + \textit{diaeresis} + \textit{breve}$  $a + \textit{breve} + \textit{diaeresis}$

Thus we get the following results when applying the algorithm. If the results compare as equal, then the originals are equivalent.

Original	Decompose	Sort	Result
$a\text{-} \textit{diaeresis} + \textit{underdot}$	$a + \textit{diaeresis} + \textit{underdot}$		$a + \textit{diaeresis} + \textit{underdot}$
$a + \textit{diaeresis} + \textit{underdot}$			$a + \textit{diaeresis} + \textit{underdot}$
$a + \textit{underdot} + \textit{diaeresis}$		$a + \textit{diaeresis} + \textit{underdot}$	$a + \textit{diaeresis} + \textit{underdot}$
$a\text{-} \textit{underdot} + \textit{diaeresis}$	$a + \textit{underdot} + \textit{diaeresis}$	$a + \textit{diaeresis} + \textit{underdot}$	$a + \textit{diaeresis} + \textit{underdot}$
$a\text{-} \textit{diaeresis} + \textit{breve}$	$a + \textit{diaeresis} + \textit{breve}$		$a + \textit{diaeresis} + \textit{breve}$
$a + \textit{diaeresis} + \textit{breve}$			$a + \textit{diaeresis} + \textit{breve}$
$a + \textit{breve} + \textit{diaeresis}$			$a + \textit{breve} + \textit{diaeresis}$
$a\text{-} \textit{breve} + \textit{diaeresis}$	$a + \textit{breve} + \textit{diaeresis}$		$a + \textit{breve} + \textit{diaeresis}$

Characters have the same priority if they interact typographically; different priorities if they do not. Enclosing characters have the priority of base characters.

 *Base characters never sort relative to one another, so the amount of work is limited by the number of non-spacing marks in a row.*

This algorithm establishes the canonical equivalence of two sequences of characters. For example, this algorithm establishes the canonical equivalence of $o + \textit{diaeresis}$ to \ddot{o} . This should not be confused with language-specific collation or matching, which may add additional information. For example, in Swedish, \ddot{o} is treated as a completely different letter from o , collated after z . In German, \ddot{o} is weakly equivalent to oe , and collated with oe . In English or French, \ddot{o} is just an o with a diacritic that indicates that it is pronounced separately from the previous letter (as in *coöperate*), and is collated with o .

Collation sequences may not require correct sorting outside of a given domain, and may not choose to invoke the canonical equivalency algorithm for excluded characters. For example, an English collator may not need to sort Cyrillic letters properly: in that case, it does not have to maximally decompose and reorder Cyrillic letters, and may just choose to sort them according to Unicode order.

4.4.1 Maximal Decomposition

The maximal decomposition contained in Appendix I: Unicode 1.1 Character List, p. 43 does contain some compatibility characters which are not in the compatibility zone. In general, compatibility characters are those that would not otherwise have been encoded, because they are in some sense variants of characters that have already been coded. The prime examples are the glyph variants in the compatibility zone: halfwidth characters, Arabic contextual form glyphs, and Arabic ligatures.

Historically, a number of such characters were added to Unicode before there was a recognized “Compatibility Zone.” Examples of these include Roman numerals, such as the *IV* “character.” By the time such a zone was distinguished, it was impractical to move those characters to the zone. However, for the companies that form the Unicode consortium, it is important to be able to identify which characters are compatibility characters so that Unicode systems can treat them in a uniform way.

Extreme care must be taken not to make artificial distinctions among characters. This is the reason, for example, that the IPA characters are identified with the Latin characters where possible. Users become very confused when they see an Å on the screen, but their search dialog does not find it—because it is *not* an Å (A-ring), it is an Å (Angstrom). Normally, many characters have different usages, such as “,” for either thousands-separator (English) or decimal-separator (French). Unicode tries to avoid duplicating characters just because of specific usage in different languages.

Identifying a character A as a compatibility variant of another character B implies that generally A can be remapped to B without loss of information other than formatting. Such remapping cannot always take place: many of the compatibility characters are in place just to allow systems to maintain one-to-one mappings to existing code sets. In such cases, a remapping would lose information that is felt to be important in the original set. A complete set of mappings is supplied in Appendix I: Unicode 1.1 Character List, p. 43, but implementations may choose to use a subset of these mappings in specific domains because of these issues.

It is important to realize that the compatibility mappings are specific to a version of Unicode. Some changes may occur as the result of character additions in the future. For example, if new precomposed characters are added, it may result in the addition of mappings between those characters and the corresponding composed character sequences. If a new non-spacing mark is added, it may produce decompositions for precomposed characters that did not previously have decompositions.

☞ *A large number of mappings are introduced by the very large number of additional characters in Unicode 1.1 from ISO/IEC 10646-1. Should errors be found in the mapping list in the future, errata notices will be made available through Unicode, Inc. and on the corresponding unicode.org FTP site.⁴*

⁴ As in any other case, you can file bug reports against the mapping list. Contact the Unicode Consortium for more information.