INTERNATIONAL STANDARD

ISO/IEC 14651

Sixth edition
2020-12

Information technology — International string ordering and comparison — Method for comparing character strings and description of the common template tailorable ordering

Technologies de l'information — Classement international et comparaison de chaînes de caractères — Méthode de comparaison de chaînes de caractères et description du modèle commun et adaptable d'ordre de classement

Reference number
ISO/IEC 14651:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 2, *Coded character sets*.

This sixth edition cancels and replaces the fifth edition (ISO/IEC 14651:2019), which has been technically revised.

The main changes compared to the previous edition are as follows:

— ordering data has been added for the new characters standardized in the sixth edition of ISO/IEC 10646 (2020);

— the content of 6.2.2 has been revised for more completeness;

— the weights of character U+A9B5 (JAVANESE VOWEL SIGN TOLONG) have been changed, as the latter is considered a variant of character U+A9B4 (JAVANESE VOWEL SIGN TARUNG). This needed to be fixed.

Any feedback or questions on this document should be directed to the user’s national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document provides a method, applicable around the world, for ordering text data, and provides a Common Template Table which, when tailored, can meet a given language's ordering requirements while retaining reasonable ordering for other scripts.

The Common Template Table requires some tailoring in different local environments. Conformance to this document requires that all deviations from the template, called "deltas", be declared to document resultant discrepancies.

This document describes a method to order text data independently of context.

ISO/IEC TR 30112 has specifications for ordering that informatively complement the specifications in this document and indicates where additional information can be sought on ordering keywords defined in this document.

Information technology — International string ordering and comparison — Method for comparing character strings and description of the common template tailorable ordering

1 Scope

This document defines the following.

- A reference comparison method. This method is applicable to two character strings to determine their collating order in a sorted list. The method can be applied to strings containing characters from the full repertoire of ISO/IEC 10646. This method is also applicable to subsets of that repertoire, such as those of the different ISO/IEC 8-bit standard character sets, or any other character set, standardized or not, to produce ordering results valid (after tailoring) for a given set of languages for each script. This method uses collation tables derived either from the Common Template Table defined in this document or from one of its tailorings. This method provides a reference format. The format is described using the Backus-Naur Form (BNF). This format is used to describe the Common Template Table. The format is used normatively *within* this document.

- A Common Template Table. A given tailoring of the Common Template Table is used by the reference comparison method. The Common Template Table describes an order for all characters encoded in the Unicode 13.0 standard,[27] included in ISO/IEC 10646:2020. It allows for a specification of a fully deterministic ordering. This table enables the specification of a string ordering adapted to local ordering rules, without requiring an implementer to have knowledge of all the different scripts already encoded in the Universal Coded Character Set (UCS).

NOTE 1 This Common Template Table is to be modified to suit the needs of a local environment. The main worldwide benefit is that, for other scripts, often no modification is required and the order will remain as consistent as possible and predictable from an international point of view.

NOTE 2 The character repertoire used in this document is equivalent to that of the Unicode Standard version 13.0[27].

- A reference name. The reference name refers to this particular version of the Common Template Table, for use as a reference when tailoring. In particular, this name implies that the table is linked to a particular stage of development of the ISO/IEC 10646 Universal coded character set.

- Requirements for a declaration of the differences (delta) between the collation table and the Common Template Table.

This document does *not* mandate the following.

- A specific comparison method; any equivalent method giving the same results is acceptable.

- A specific format for describing or tailoring tables in a given implementation.

- Specific symbols to be used by implementations, except for the name of the Common Template Table.

- Any specific user interface for choosing options.

- Any specific internal format for intermediate keys used when comparing, nor for the table used. The use of numeric keys is not mandated either.

- A context-dependent ordering.

- Any particular preparation of character strings prior to comparison.

NOTE 1 It is normally necessary to do preparation of character strings prior to comparison even if it is not prescribed by this document (see Annex C).

NOTE 2 Annex D describes problems that gave way to this International Standard with their anticipated solutions.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 10646:2020, *Information technology — Universal Coded Character Set (UCS)*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— IEC Electropedia: available at http://www.electropedia.org/

— ISO Online browsing platform: available at https://www.iso.org/obp

3.1
character string
sequence of characters considered as a single object

Note 1 to entry: Note to entry: A character string to be ordered does not normally include the characters that delimit it, as for example an “end of line” control character in a text file to be sorted.

3.2
collating symbol
symbol (3.12) used to specify weights assigned to a *collating element* (3.4)

3.3
collation table
weighting table
mapping from *collating elements* (3.4) to *weighting elements* (3.14)

3.4
collating element
sequence of one or more characters that are considered a single entity for *ordering* (3.7)

3.5
delta
list of the differences between a given *collation table* (3.3) and another one

Note 1 to entry: The given collation table, together with a given delta, forms a new collation table.

Note 2 to entry: Unless otherwise specified in this document, the term “delta” always refers to differences from the Common Template Table as defined in this document.

3.6
level
collation level
sequence number for a *subkey* (3.11) in the series of subkeys forming a key

3.7
ordering
collation
process by which, given two strings, it is determined whether the first one is less than, equal to, or greater than the second one

3.8
ordering key
sequence of *subkeys* (3.11) used to determine an order

3.9
preparation
collation preparation
process in which given *character strings* (3.1) are mapped to (other) character strings before the calculation of the *ordering key* (3.8) for each of the strings

3.10
reference comparison method
method for establishing an order between two *ordering keys* (3.8)

Note 1 to entry: See Clause 6.

3.11
subkey
sequence of weights computed for a *character string* (3.1)

3.12
symbol
collating element (3.4)

3.13
weight
collation weight
positive integer value, used in *subkeys* (3.11), reflecting the relative order of *collating elements* (3.4)

3.14
weighting element
list of a given number of weights sequentially ordered by level

4 Symbols and conventions

Following ISO/IEC 10646, characters are referenced as U*X* where *X* stands for a series of one to eight hexadecimal digits (where all the letters in the hexadecimal string are in upper case) and refers to the value of that character in ISO/IEC 10646. This convention is used throughout this document.

In the Common Template Table, arbitrary symbols representing weights are used according to the BNF notation description in 6.3.1.

5 Conformance

A process is conformant to this document if it produces results identical to those that result from the application of the specifications given in 6.2 to 6.5.

A declaration of conformity to this document shall be accompanied by a statement, either directly or by reference, of the following:

— the number of levels that the process supports; this number shall be at least three;

— whether the process supports the forward position processing parameter;

- whether the process supports the backward processing parameter and at which level;
- the tailoring *delta* described in 6.4 and how many levels are defined in the delta;
- if a preparation process is used, the method used shall be declared.

It is the responsibility of implementers to show how their delta declaration is related to the table syntax described in 6.3, and how the comparison method they use, if different from the one mentioned in Clause 6, can be considered as giving the same results as those prescribed by the method specified in Clause 6. The use of a preparation process is optional and its details are not specified in this document.

It is strongly recommended that processes present available tailoring options to users.

6 String comparison

6.1 Preparation of character strings prior to comparison

It can be necessary to transform character strings before the reference comparison method is applied to them. Although not part of the Scope of this document, preparation can be an important part of the ordering process. See Annex C for some examples of preparation.

Characters of the input string shall be encoded according to ISO/IEC 10646 (UCS) or a mapping to ISO/IEC 10646 shall be provided if another encoding scheme is used.

Therefore it can be an important part of the preparation phase to map characters from a non-UCS encoding scheme to the UCS for input to the comparison method. This task can, amongst other things, encompass the correct handling of escape sequences in the originating encoding scheme, the mapping of characters without an allocated UCS codepoint to an application-defined codepoint in the private zone area and change the sequence of characters in strings that are not stored in logical order. For example, for visual order Arabic code sets, input strings shall be put into logical order; and for some bibliographic code sets, strings with combining accents stored before their respective base character require that the combining accents be put after their base character. The resulting string sequence may then have to be remapped into its original encoding scheme.

The Common Template Table is designed so that combining sequences and corresponding single characters (precomposed) will have precisely the same ordering. To avoid inadvertently breaking this invariant (and in the process breaking Unicode conformance), tailoring should reorder combining sequences when corresponding precomposed characters are reordered. For example, if Ä is reordered after Z, then the sequence <A>+<combining diaeresis> should also be reordered. To avoid exposing encoding differences that can be invisible to the end-user, it is recommended that strings be normalized according to Unicode normalization form NFD to achieve this equivalence (see[29]).

Escape sequences and control characters constitute very sensitive data to interpret, and it is highly recommended that preparation should filter out or transform these sequences.

NOTE Since the reference method is a logical statement for the mechanism for string comparison, it does not preclude an implementation from using a non-UCS character encoding only, as long as it produces results as if it were using the reference comparison method.

6.2 Key building and comparison

6.2.1 Preliminary considerations

6.2.1.1 Assumptions

The collation table is a mapping from collating elements to weighting elements. In each weighting element, four levels are described in the Common Template Table. This number of levels can be extended or reduced, but cannot be less than 3, in tailoring.

NOTE In the Common Template Table, levels generally have the following characteristics, although this purpose is not absolute.

Level 1: This level generally corresponds to the set of common letters of the alphabets for that script, if the script is alphabetic, and to the set of common characters of the script if the script is ideographic or syllabic.

Level 2: This level generally corresponds to diacritical marks affecting each basic character of the script. For some languages, letters with diacritics are always considered an integral part of the basic letters of the alphabet and are not considered at this second level, but rather at the first. For example, in Spanish, N TILDE is considered a basic letter of the Latin script. Therefore, tailoring for Spanish will change the definition of N TILDE from "the weight of an N in the first level and the weight of a TILDE in the second level" to "the weight of an N TILDE (placed after N and before O) in the first level, and indication of the absence of a diacritic in the second level". For some characters, variant letter shapes are also dealt with on level 2. An example of this is ß, the LATIN SMALL LETTER SHARP S, which is treated as equivalent to ss on level 1, but traditionally distinguished from it on level 2.

Level 3: This level generally corresponds to case distinctions (upper and lower case) or to distinctions based on variant letter shapes (like the distinction between Hiragana and Katakana).

Level 4: This level generally corresponds to weighting differences that are less significant than those at the other levels. Often the last level (level 4 in the Common Template Table) is intended to specify additional weighting for "special" characters, i.e. characters normally not part of the spelling of words of a language (such as dingbats, punctuation, etc.), sometimes called "ignorable" characters in the context of computerized ordering.

6.2.1.2 Processing properties

A given tailored table has specific scanning and ordering properties. These properties can have been changed by the tailoring.

A scanning direction (forward or backward) for each level is used to indicate how to process the string. The scanning direction is a global property of each level defined in the tailored table.

If the last level is greater than three, there is an optional property of this level of comparison called “position” option: when active, a comparison on the numeric position of each “ignorable” character in the two strings is effected, before comparing their weights. In other words, for two strings equivalent at all levels except the last one, the string having an ignorable character in the lowest position comes before the other one. In case corresponding ignorable characters are at the same position, then their weights are considered, until a difference is found. *Support* for this kind of processing is *optional* and is not necessary to claim conformance to this document.

NOTE The scanning direction (forward or backward) is not normally related to the natural writing direction of scripts. The scanning direction applies to the logical sequence of the coded character string.

According to ISO/IEC 10646, for scripts written right to left, such as Arabic, the first characters in the logical sequence correspond to the rightmost characters in their natural presentation sequence. Conversely, for the Latin script, written left to right, the first characters in the logical sequence correspond to the leftmost characters in their natural presentation sequence.

Scanning forward starts with the lowest position in the logical sequence, while scanning backward starts from the highest position, independently of the presentation sequence. The scanning direction for ordering purposes is a global property of each level described in the table.

In ISO/IEC 10646, the Arabic script is artificially separated into two pseudo-scripts: 1) the logical, intrinsic Arabic, coded independently of contextual shapes, and 2) the Arabic presentation forms. Both allow the complete coding of Arabic, but intrinsic Arabic is normally preferred for better processing, while presentation-form Arabic is preferred by some presentation-oriented applications. ISO/IEC 10646 does not prescribe that the presentation forms be stored in any specific order, and in some implementations, the storage order for the latter is the reverse of the storage order used for intrinsic Arabic. It is therefore advisable that the preparation phase be used to make sure that Arabic presentation forms and other Arabic characters be fed to the comparison method in logical order.

A tailored sort table can be separated into sections for ease of tailoring. Each section is then assigned a name consistent with the specification in 6.3.1. One of the tailoring possibilities is to assign a given order to each section and to change the relative order of an entire section relative to other sections.

6.2.2 Reference ordering key formation

6.2.2.1 General

When two strings are to be compared to determine their relative order, the two strings are first parsed into a sequence of collating elements taking into account the multi-character "collating-element" statements declared and used in a tailored table (*if* the syntax of 6.3.2 is used). For the syntax used for expressing the Common Template Table, the name of a collating element consisting of a single character, is formed by the UCS value of the character, expressed as a hexadecimal string, prefixed with "U". For multi-character collating elements, the name and association to characters can be found via the collating elements declarations.

NOTE Collating elements with more characters have preference over shorter ones. As an example, if a multicharacter collating element is defined for "abc" and another one is defined for "ab" or for "bc", then if "abc" is encountered, the collating element for "abc" will apply and not the one for "ab" or "bc".

Then, a sequence of *m* intermediary subkeys is formed out of a character string, where *m* is the number of levels described in a tailored collation weighting table.

Each ordering key is a sequence of subkeys. Each subkey is a list of numeric weights. A subkey is formed by successively appending the list of the weights assigned, at the level of the subkey, to each collating element of the string. The keyword "IGNORE" in the Common Template Table at the place of a sequence of collating symbols at a level indicates that the sequence of weights at that level for that collating element is an empty sequence of weights.

6.2.2.2 Weighting elements to be ignored

When forming a sort key, collating elements ignored at the first level or at the two first levels and that follow a collation element ignored at all levels but the last one, do not keep their weights as defined in the common template table (or a tailored table); each of these weights shall be zeroed (this means that "IGNORE" shall be assigned to each non-nil weight).

6.2.2.3 Implicit weights computing

If there is no entry in the tailored table for a character of the input string, then the character's weights are undefined. In this case, one shall compute an "implicit" primary weight consisting of a pair of 16-bit words — call them "aaaa" and "bbbb" — and assume that lines like the following were added to the weighting table:

<UXXXX> "<R{aaaa$_{16}$}><T{bbbb$_{16}$}>";<BASE>;<MIN>;<SFFFF>

NOTE <SFFFF> (at the last level) is the largest level 1 weight in the Common Template Table.

Distinctions are needed among characters with no entry in the weighting table, and implicit primary weights are thus computed as defined here:

a) Unified Han ideographs:

For a given Han character at code point *cp*:

base_weight = 0xFB40 for original URO

base_weight = 0xFB80 for Extension A through Extension G Han characters

aaaa = [base_weight + (cp >> 15)]$_{16}$

bbbb = [(cp & 0x7FFF) | 0x8000]$_{16}$

b) Tangut ideographic and component characters:

For a given Tangut character at code point *cp*:

base_weight = 0xFB00

aaaa = [base_weight]$_{16}$

bbbb = [(cp - 0x17000) | 0x8000]$_{16}$

c) Nüshu ideographic characters:

For a given Nüshu character at code point *cp*:

base_weight = 0xFB01

aaaa = [base_weight]$_{16}$

bbbb = [(cp - 0x1B170) | 0x8000]$_{16}$

d) Khitan small script ideographic characters:

For a given Khitan small script character at code point *cp*:

base_weight = 0xFB02

aaaa = [base_weight]$_{16}$

bbbb = [(cp - 0x18B00) | 0x8000]$_{16}$

e) Any other code point not mentioned in the table (non-characters, surrogates, and any characters that are not assigned in the current repertoire):

For a given character at code point *cp*:

base_weight = 0xFBC0

aaaa = [base_weight + (cp >> 15)]$_{16}$

bbbb = [(cp & 0x7FFF) | 0x8000]$_{16}$

Thus, the value of the code point is used to calculate the desired weights, by operating on individual bits. The two numbers, converted to four-character hexadecimal values, will then take the following form: "<Raaaa><Tbbbb>".

Decomposable characters are excluded from these computings, as they have an entry in the Common Template Table (with dual primary weights).

If a string contains ill-formed code unit sequences, two approaches show up: the sequence can be handled as if it were U+FFFD (REPLACEMENT CHARACTER), or each subsequence can be ignored. The same approaches can be adopted for any out-of-range values (cp > $10FFFF_{16}$).

NOTE 1 Areas in which Han, Tangut, Nüshu and Khitan small script characters are found are defined in the comment lines added to the end of the Common Template Table.

NOTE 2 When computed weights are used, characters without explicit mappings are sorted in code point order within each set they belong to (Han, Tangut, Nüshu, Khitan small script, others) and they sort properly with respect to the rest of the characters.

6.2.2.4 Subkeys formation

There are three ways of forming subkeys: subkeys formed using the "forward" processing parameter, subkeys formed using the "backward" processing parameter, and subkeys formed using the "forward,position" processing parameter. Subkeys that use the "position" option can only occur at the last level, and only if that level is greater than three. Support of the "position" option is not required for conformance. If the processing parameter "forward,position" is not supported, "forward,position" shall be interpreted as if the processing parameter had been "forward".

6.2.2.5 Formation of subkeys for the first three levels

With the "forward" or "forward,position" processing parameter, during (forward) scanning of each collating element of the input character string, one or more weights are obtained. These weights are obtained by matching the collating element in the given tailored collation weighting table, obtaining the list of weights assigned to the collating element at the particular level. The obtained weight list is appended to the end of the subkey.

Subkeys, at a particular level, formed with the "backward" level processing parameter are built by forming a subkey as with the "forward" parameter, then reversing that subkey weight by weight.

6.2.2.6 Formation of the level 4 subkey

At the last level, the subkey is built as described in the preceding paragraph. However, once the ordering key is completely formed (when the end of the string is reached), there are two possible approaches:

a) With the "forward" processing parameter: *all* the <SFFFF> symbols are backed out of the subkey;

b) With the "forward,position" processing parameter: any *trailing* sequence of <SFFFF> symbol(s) shall be removed from the subkey (leaving intact the remaining part of the subkey).

6.2.3 Reference comparison method for ordering character strings

The reference comparison method for ordering two given character strings (*after* collation preparation, which is not part of the reference comparison method itself) is to compare ordering keys generated by the reference key formation method described in 6.2.2.

— Begin by building an ordering key, using a given tailored collation weighting table, for each of the two given character strings being compared.

— Then compare the resulting keys according to the key ordering definition presented in 6.2.4. Keys can be compared either up to a given level, or up to the last level of the given tailored collation weighting table.

NOTE The comparison can be made *while* generating the ordering keys for two strings to be compared, stopping the key generation when the order of the strings can be determined. Such a technique is sometimes termed lazy evaluation, and some systems support it by default. This avoids generating the full ordering key when an ordering difference is found early in the keys. When a bigger set of strings are to be ordered, one can generate the ordering keys, for example, and store each key or an initial segment of each, before comparing the keys.

6.2.4 Key ordering definition

Weights for different levels should not be compared, which implies that subkeys at different levels should not be compared, nor should keys generated from different tailored tables be compared.

NOTE 1 This allows implementations to assign weightings at each level independently of the other levels, and independently of other tailorings.

m is the maximal level of a given tailored table. Recall that a key is a list, of length m, of subkeys; a subkey is a list of weights; and a weight is a positive integer. Other notations used below are the following:

— L_z is the length of the subkey z, i.e., the number of weights in the subkey;

— $z_{\mathrm{wt(a)}}$, where $1 \leq a \leq L_z$, is the weight at index position a (an integer > 0) of the subkey z;

— $u_{sk\mathrm{(b)}}$, where $1 \leq b \leq m$, is the subkey at level b (an integer > 0) of the key u.

The orderings of weights, subkeys, and ordering keys (up to a given level, or up to the last level) are total order relations, defined for a given tailored collation table as follows.

a) Weights are positive integers (in the reference method) and are compared as such for the purposes of collation.

b) A subkey v is *less than* a subkey w (written $v < w$) **if and only if** there exists an integer, i, where $1 \leq i \leq L_v + 1$ and $i \leq L_w$, such that

   — $i = 1$ and $v_{\mathbf{wt}(i)} < w_{\mathbf{wt}(i)}$, or

   — for all integers, j, where $1 \leq j < i$, the equality $v_{\mathbf{wt}(j)} = w_{\mathbf{wt}(j)}$ holds, and either

      — $i \leq L_v$ and $v_{\mathbf{wt}(i)} < w_{\mathbf{wt}(i)}$, or

      — $i = L_v+1$ and $0 < w_{\mathbf{wt}(i)}$.

   A subkey v is *greater than* a subkey w (written $v > w$) **if and only if** w is less than v. A subkey v is *equal to* a subkey w (written $v = w$) **if and only if** neither v is less than w, nor w is less than v.

c) An ordering key x is *less than* an ordering key y at level s (written $x <_s y$) **if and only if** there exists an integer i, where $1 \leq i \leq s$ and $i \leq m$, such that

   — $i = 1$ and $x_{\mathbf{sk}(i)} < y_{\mathbf{sk}(i)}$, or

   — for all integers j, where $1 \leq j < i$, the equality $x_{\mathbf{sk}(j)} = y_{\mathbf{sk}(j)}$ holds, and $x_{\mathbf{sk}(i)} < y_{\mathbf{sk}(i)}$.

   An ordering key x is *greater than* an ordering key y at level s (written $x >_s y$) **if and only if** y is less than x at level s. An ordering key x is *equal to* an ordering key y at level s (written $x =_s y$) **if and only if** neither x is less than y at level s, nor y is less than x at level s.

d) For ordering keys, <, >, and = are defined as $<_m$, $>_m$, and $=_m$ respectively.

NOTE 2 For ordering keys, $x <_t y$ implies $x <_{t+1} y$, $x >_t y$ implies $x >_{t+1} y$, $x =_t y$ implies $x =_{t-1} y$, $x <_0 y$ is false, $x >_0 y$ is false, and $x =_0 y$ is true. Above level m, for a given tailored table, there are no further ordering distinctions. Note that this definition implies that if two ordering keys are in the 'less than' relationship at level 1, they will also be in the 'less than' relationship at levels 2, 3, 4, etc. In general, whenever two ordering keys are less than at a given level, they will also automatically be less than at all subsequent, higher levels. Conversely, if two ordering keys are equal at a given level, they will also automatically be equal at all preceding, lower levels.

NOTE 3 The decomposition of character string into ordering keys of different levels of comparison from the most significant to the least significant also facilitates fuzzy searches and searches for equivalences. Annex E documents this kind of comparison.

6.3 Common Template Table: Formation and interpretation

6.3.1 General

This subclause specifies the following:

- the syntax used to form the Common Template Table in Annex A or a tailored table based upon the Common Template Table as expressed in Annex A;
- conditions of well-formedness of a table using this syntax;
- interpretation of tailoring statements in deltas for tables formed using this syntax;
- evaluation from symbols to weights of tailored tables formed using this syntax;
- conditions for considering two tables as equivalent;
- conditions for considering comparison results as equivalent.

6.3.2 BNF syntax rules for the Common Template Table in Annex A

6.3.2.1 General

Definitions between <angle brackets> make use of terms not defined in this BNF syntax and assume general English usage.

Other conventions:

*	indicates 0 or more repetitions of a token or a group of tokens;
+	indicates 1 or more repetitions of a token or a group of tokens;
?	indicates optional occurrence of a token or a group of tokens (0 or 1 occurrences);
	parentheses are used to group tokens;
	production rules are terminated by a semicolon.

Define collation tables as sequences of lines:

```
weight_table = common_template_table | tailored_table ;
common_template_table =
                 simple_line+ ;
tailored_table = table_line+ ;
```

Define the line types:

```
simple_line =     (symbol_definition | collating_element |
                  weight_assignment | order_end)?
                  line_completion ;
table_line =      simple_line | tailoring_line ;
tailoring_line =  (reorder_after | order_start | reorder_end |
                  section_definition | reorder_section_after)
                  line_completion ;
```

Define the basic syntax for collation weighting:

```
symbol_definition =
                 ’collating-symbol’ space+ symbol_element ;
symbol_element = symbol | symbol_range ;
symbol_range =   symbol ’..’ symbol ;
symbol =         simple_symbol | ucs_symbol ;
ucs_symbol =     (’<U’  one_to_eight_digit_hex_string  ’>’) |
                 (’<U-’ one_to_eight_digit_hex_string ’>’) ;
simple_symbol =  ’<’ identifier ’>’ ;
collating_element =
                 ’collating-element’ space+ symbol space+
                 ’from’ space+ quoted_symbol_sequence ;
quoted_symbol_sequence =
                 ’"’ simple_weight+ ’"’ ;
weight_assignment =
                 simple_weight | symbol_weight ;
simple_weight =  symbol_element | ’UNDEFINED’ ;
symbol_weight =  symbol_element space+ weight_list ;
weight_list =    level_token (semicolon level_token)* ;
level_token =    symbol_group | ’IGNORE’ ;
symbol_group =   symbol_element | quoted_symbol_sequence ;
order_end =      ’order_end’ ;
```

Define the tailoring syntax:

```
reorder_after =  ’reorder-after’ space+ target_symbol ;
target_symbol =  symbol ;
order_start =    ’order_start’ space+ multiple_level_direction ;
multiple_level_direction =
                 (direction semicolon)* direction (’,position’)? ;
direction =      ’forward’ | ’backward’ ;
reorder_end =    ’reorder-end’ ;
section_definition =
                 section_definition_simple |
                 section_definition_list ;
section_definition_simple =
                 ’section’ space+ section_identifier ;
section_identifier =
                 identifier ;
section_definition_list =
                 ’section’ space+ section_identifier space+
                 symbol_list ;
symbol_list =    symbol_element (semicolon symbol_element)* ;
reorder_section_after =
                 ’reorder-section-after’ space+ section_identifier
                 space+ target_symbol ;
```

Define low-level tokens used by the rest of the syntax:

```
identifier =    (letter | digit) id_part* ;
id_part =       letter | digit | ’-’ | ’_’ ;
line_completion =
                space* comment? EOL ;
comment =       comment_char character* ;
one_to_eight_digit_hex_string =
                hex_upper |   hex_upper hex_upper |
                hex_upper hex_upper hex_upper |
                hex_upper hex_upper hex_upper hex_upper |
                hex_upper hex_upper hex_upper hex_upper hex_upper |
                hex_upper hex_upper hex_upper
                hex_upper hex_upper hex_upper |
                hex_upper hex_upper hex_upper
                hex_upper hex_upper hex_upper hex_upper |hex_upper
                hex_upper hex_upper hex_upper
                hex_upper hex_upper hex_upper hex_upper ;
hex_numeric_string =
                hex_upper+ ;
space =         ’ ’ | <TAB> ;
semicolon =     ’;’ ;
comment_char = ’%’ ;
digit =         ’0’ | ’1’ | ’2’ | ’3’ | ’4’ |
                ’5’ | ’6’ | ’7’ | ’8’ | ’9’ ;
hex_upper =     digit | ’A’ | ’B’ | ’C’ | ’D’ | ’E’ | ’F’ ;
letter =        ’a’ | ’b’ | ’c’ | ’d’ | ’e’ | ’f’ | ’g’ |
                ’h’ | ’i’ | ’j’ | ’k’ | ’l’ | ’m’ | ’n’ |
                ’o’ | ’p’ | ’q’ | ’r’ | ’s’ | ’t’ | ’u’ |
                ’v’ | ’w’ | ’x’ | ’y’ | ’z’ |
                ’A’ | ’B’ | ’C’ | ’D’ | ’E’ | ’F’ | ’G’ |
                ’H’ | ’I’ | ’J’ | ’K’ | ’L’ | ’M’ | ’N’ |
                ’O’ | ’P’ | ’Q’ | ’R’ | ’S’ | ’T’ | ’U’ |
                ’V’ | ’W’ | ’X’ | ’Y’ | ’Z’ ;
EOL =           <end-of-line in the text conventions in use> ;
character =     <any member of the repertoire of the encoded
```

```
                    character set in use, not including any
                    characters used to delimit the end of lines> ;
```

6.3.2.2 Keyword usage

The usage of the following locale syntax keywords is as follows.

collating-symbol Defines a collating symbol representing a weight.

collating-element Defines a collating element symbol representing a multi-character collating element.

order_start Defines collation rules. This statement is (after reordering is done) followed by one or more collation order statements, assigning multi-level collation weightings to collating elements. A collating element is either a character or a defined substring.

order_end Specifies the end of the collating order statements.

reorder-after Redefines collating rules. Specifies after which collating symbol weighting the lines between the "reorder-after" and the next (or "reorder-end") are to be moved. This statement is followed by one or more collation table lines. The result is to reassign collating symbol's values or collating element's weightings.

reorder-end Specifies the end of the "reorder-after" collating order statements.

section Defines a section of the table. A section can be moved as a whole by "reorder-section-after".

reorder-section-after Redefines the order of sections. This statement is followed by a section symbol and a target symbol. The result is to reassign collating symbol's values or collating element's weightings.

6.3.3 Well-formedness conditions

WF 1. Any *simple_symbol* occurring in a *weight_list* shall also occur in the initial *symbol_element* of a *symbol_weight*, or *symbol_definition* that occurs no later in the sequence of lines that constitute a *weight_table*.

All *simple_symbols* shall be "defined" before they are "used".

WF 2. No *symbol* that occurs in a *symbol_definition* in a *weight_table* that contains no *tailoring_lines* may occur in another *symbol_definition* in the same *weight_table*.

Duplication of collation weighting symbols is prohibited. This is true for the Common Template Table itself. It shall remain true for a *tailored_table* after all reordering of lines has been applied.

WF 3. All *weight_lists* in a *tailored_table* shall contain the same number of *level_tokens*. An empty *level_token* shall be interpreted as the *collating_element* itself.

A tailorable table shall be consistent in its use of levels throughout.

WF 4. A tailored_table shall contain exactly one order_start statement followed later by exactly one order_end statement. The order_start statement shall appear after the symbol_definition entries and before the symbol_weight entries after all reordering of lines has been applied.

WF 5. A multiple_level_direction in a tailored_table shall contain the same number of directions as the number of level_tokens of any weight_list in that tailored_table.

Any *order_start* shall have the same number of levels as is generally used in the table.

WF 6. If a *level_token* in a *weight_list* consists of a *symbol_group*, all successive *level_tokens* in that *weight_list* shall also consist of a *symbol_group*.

IGNORE shall not be used at a level after an explicit symbol for a weighting.

WF 7. Any *section_identifier* occurring in a *reorder_section_after* shall occur in a *section_definition* that occurs earlier in the sequence of *table_lines* that constitutes a *tailored_table*.

All *section_identifiers* shall be “defined” before they are “used”.

WF 8. No two *section_definitions* in a *tailored_table* shall contain the same values in their *section_identifiers*.

Multiple definitions of *sections* are prohibited; *section_identifiers* shall be unique.

WF 9. Each *reorder_after* in a *tailored_table* shall be followed at a later point in that tailored table by a *reorder_end* or another *reorder_after*.

WF 10. If the *section_identifier* of a statement *reorder_section_after* corresponds to the *section_identifier* of a *section_definition_list*, then the *target_symbol* of that statement *reorder_section_after* shall not have the same value as any of the *symbols* of the *section_definition_list*.

A *section* shall not be reordered after a line which the *section* itself contains; recursive relocation of lines is prohibited.

WF 11. Any *symbol_range* shall contain two *symbols* which meet the following conditions. 1) Each of the two *symbols* shall contain a common prefix of length one. The prefix can be any letter except 'U. 2) The portions of *identifier* of each of the two *symbols* following the common prefix shall be a *hex_numeric_string* containing the same number of hexadecimal digits. 3) When interpreted as numeric values, the *hex_numeric_string* of the first *symbol* shall be less than the *hex_numeric_string* of the second *symbol*. One plus the positive integral difference between the *hex_numeric_string* of the second *symbol*, interpreted as a numeric value, and the *hex_numeric_string* of the first *symbol*, interpreted as a numeric value, constitutes the number of *values* of the *symbol_range*.

NOTE A well-formed *symbol_range* is of a form such as <S4E00>..<S9FA5>, where the common prefix is “S”, and the rest of the *identifier* portion of each *symbol* is a *hex_numeric_string*.

WF 12. Any *symbol_weight* that contains more than one *symbol_range* shall contain only *symbol_ranges* that meet the following requirement. Each *symbol_range* following the first *symbol_range* shall have the same number of values in its *range* as that of the first *symbol_range*.

NOTE This condition guarantees that all expanded ranges will be well-formed, since for any one *symbol_weight*, all of the range expansions will have the same number of values.

6.3.4 Interpretation of tailored tables

I 1. A *section* consists either 1) of the list of *simple_lines* which contain a *symbol_definition* whose value is equal to any *symbol* contained in the *symbol_list* in a *section_definition_list*, or 2) of the list of *simple_lines* following a *section_definition_simple* in a *tailored_table*.

NOTE A *section* is defined 1) by a specific *symbol_list*, or 2) by taking all the lines following the *section_definition* until another tailoring line such as an *order_start*, a *reorder_section_after*, another *section_definition*, or the end of the table is encountered.

I 2. A *simple_line* consisting of a *symbol_definition* containing a *symbol_range* is equivalent to a sequence of *simple_lines*, where each of those lines contain a *symbol* in place of the *symbol_range*. The *symbol* for each successive *simple_line* is generated by concatenating a *hex_numeric_string* to the common prefix of the *symbol_range,* in numeric order, starting with the hex value associated with the *hex_numeric_string* of the first *symbol* the range, and ending with the hex value associated with the *hex_numeric_string* of the second *symbol*. The *hex_numeric_string* concatenated to the common prefixes shall contain the same number of digits as the *hex_numeric_string* of the first symbol. The number of *simple_lines* thus generated is equal to the number of symbols in the *symbol_range*.

NOTE A symbol_definition of the form "collating-symbol <S0301>..<S0303>" is equivalent to the three lines:

— collating-symbol <S0301>;

— collating-symbol <S0302>;

— collating-symbol <S0303>.

I 3. A *simple_line* consisting of a *symbol_weight* containing one or more *symbol_ranges* is equivalent to a sequence of *simple_lines*, where each *symbol_range* has been expanded into a sequence of *symbols*, as described in I 2 for *symbol_definition*

NOTE A symbol_weight of the form "<U2000>..<U2002> <S0301>..<S0303>;<BASE>;<MIN>;<SFFFF>" is equivalent to the three lines:

— <U2000> <S0301>;<BASE>;<MIN>;<SFFFF>;

— <U2001> <S0302>;<BASE >;<MIN>;<SFFFF>;

— <U2002> <S0303>;<BASE >;<MIN>;<SFFFF>.

I 4a. A *tailored_table* containing a *reorder_after* is equivalent to the *tailored_table* where:

a) all *table_lines* that were ahead of the *reorder_after* and that contained *weight_assignments* whose initial *symbol* matches the initial *symbol* of any *weight_assignment* in the *table_lines* between the *reorder_after* and *reorder_end* have been removed,

b) the *table_lines* between that *reorder_after* and the first subsequent *reorder_end* to immediately follow the first *table_line* in the *tailored_table* containing a *weight_assignment* whose initial *symbol* is the same as the *target_symbol* in the *reorder_after* have been reordered, and

c) that *reorder_after* and that *reorder_end* have been removed.

Move the block of lines between the *reorder_after* and the *reorder_end* to follow the *target_symbol*, delete any prior lines that duplicate the *symbol_definitions* of the reordered lines, and remove the *reorder_after* and *reorder_end* themselves.

I 4b. When a *tailored_table* contains multiple groups of lines to be reordered, the table is interpreted by processing each *reorder_after* sequentially, starting from the first line of the table.

NOTE Subsequent line reorderings can impact lines that themselves were reordered by prior reorderings.

I 5. A *tailored_table* containing a *reorder_section_after* is equivalent to the *tailored_table* with the *section* associated with that reorder_*section_after* reordered (in the same relative order as the *table_lines* have in that *section*) to immediately follow the last *table_line* in the *tailored_table* containing a *symbol_definition* whose *symbol* is the same as the *target_symbol* in the *reorder_section_after*, and with that *reorder_section_after* removed.

I 6. A *weight_table* is said to be in normal form when it contains no *reorder_after*s nor *reorder_section_afters*.

NOTE A *tailored_table* can be put into normal form by the operations implied by I 4 and I 5.

6.3.5 Evaluation of weight tables

E 1. A *weight_table* in normal form is said to be evaluated when each *weight_assignment* in the *weight_table* is mapped to a positive integer value (a weight) such that those values increase monotonically by the order in which the *weight_assignments* occur in the *weight_table*.

NOTE 1 The *table_lines* of the *weight_table* can first be mapped to the set of positive integers, by sequential order in the table. This mapping defines an ordered set of line numbers. The *weight_assignments* are then mapped to a set of positive integers (weights) that varies monotonically with the set of line numbers.

This does not restrict the starting number for the weight of the first *weight_assignment* (other than it shall be positive) nor does it require that the numbers for these weights be immediately consecutive.

E 2. An evaluated *weight_table* is said to be collating-element-weighted when each *simple_symbol* occurring in each *weight_list* in that evaluated *weight_table* has been mapped to the weight that corresponds to the *weight_assignment* that contains the same *simple_symbol*.

NOTE 2 Each *weight_list* can be interpreted as containing either *symbol*’s mapped to integral weight values or as instances of the string ‘IGNORE’, which denotes the empty sequence of weights. At this point, the mathematical injection of strings can be defined using the *weight_table*.

NOTE 3 In a tailored table, the value of any *hex_numeric_string* associated with a *symbol* generally does not reflect the numeric weighting of the *symbol*.

6.3.6 Conditions for considering specific table equivalences

A *weight_table* TBL1 and a *weight_table* TBL2 are said to be equivalent at a particular level if any comparison of strings using those tables up to that level results in the same ordering.

When two tables are claimed to be equivalent, one shall always get the same results in forming keys for two strings based on TBL1 and comparing them than by forming keys based on TBL2 and comparing them.

6.3.7 Conditions for results to be considered equivalent

An implementation of international string ordering is conformant with this document if, for any set of strings *S* defined on a repertoire *R*, the implementation can duplicate the same comparisons as those resulting from comparison of the numbers from an injection constructed according to the rules of 6.2.3.

6.4 Declaration of a delta

Tailoring shall be based upon the Common Template Table described in Annex A. Tailoring can be accomplished using any syntax that is equivalent to the one described in this document. Examples of tailoring delta can be found in Annex B.

NOTE 1 For example, ISO/IEC TR 30112, uses a compatible extension of the syntax used in this document for tailoring. A tailoring delta can also be expressed using the syntax of the Unicode collation algorithm (see[27]). It has also been demonstrated that a tailoring delta can also be expressed using an XML-conformant mark-up scheme.

Any declaration of conformance to this document shall be accompanied with a declaration of the differences between the collation weighting table and the Common Template Table. A delta shall contain the equivalent of the following:

a) at least one valid *order_start* entry described in 6.3.2; an unlimited number of sections containing an *order_start* entry and an *order_end* entry can be declared;

b) the number of levels used for comparison;

c) the list of *symbol_definition* weights (as defined in 6.3.2) added and the position of the *symbol_definition* entry after which each insertion is made;

d) the list of *simple_line* entries (as defined in 6.3.2) deleted or inserted, referencing the position of the *simple_line* entry in the Common Template Table after which the insertions are made.

It is recommended that a delta should not be bigger than necessary.

In cases where a process allows the end-user to tailor the table himself or herself, a statement of conformance shall indicate which of the four elements of the previous list are tailorable and which of those four elements are not tailorable. For those which are not tailorable, the delta between fixed elements and the Common Template Table shall be declared.

NOTE 2 The declaration can use a different syntax from the one specified in 6.3 provided that the relationship with this syntax can be reasonably established. For example, the following declarations are valid:

— "Collate U00E5 after U00FE at the primary level;

— Collate U00E4 after U00E5 at the primary level. ";

— "The primary alphabet order is modified so that in all cases z < þ < å < ä".

Note that the letters å and ä are sorted after Icelandic letter thorn (þ), itself already coming after all the variants of the letter z, i.e. å and ä have a weight value higher at level 1 than the one for þ, which itself comes after the ones for all variants of z.

The above two informal expressions can reasonably be considered to be equivalent to the following more precise expression (which also gives weights at levels 2 and 3 and explicitly takes care of accented å's, accented ä's and the Ångström sign):

reorder-after <S00FE> *% Weight for THORN (after z; and unlike z, þ has no variants).*

% Declare new collation symbols (weight names):

collating-symbol <S00E5> *% for å*

collating-symbol <S00E4> *% for ä*

% Declare new collating elements for the decompositions (substring names):

collating-element <U0061_030A> **from** "<U0061><U030A>" *% decomposition of å*

collating-element <U0041_030A> **from** "<U0041><U030A>" *% decomposition of Å*

collating-element <U0061_0308> **from** "<U0061><U0308>" *% decomposition of ä*

collating-element <U0041_0308> **from** "<U0041><U0308>" *% decomposition of Ä*

% Assign weights to the new collation symbols (after THORN):

<S00E5> ***% for å***

<S00E4> ***% for ä***

reorder-end

reorder-after <SFFFF> *% The only place where we can put the order_start line.*

order_start forward;forward;forward;forward

% Use the new weighted collation symbols and collating elements to tailor the collation rules:

*% The letter **Å**:*

<U00E5> <S00E5>;<BASE>;<MIN>;<SFFFF> *% LATIN SMALL LETTER A WITH RING ABOVE*

<U0061_030A> <S00E5>;<BASE>;<MIN>;<SFFFF> *% decomposition of å*

<U00C5> <S00E5>;<BASE>;<CAP>;<SFFFF> *% LATIN CAPITAL LETTER A WITH RING ABOVE*

<U0041_030A> <S00E5>;<BASE>;<CAP>;<SFFFF> *% decomposition of Å*

<U212B> <S00E5>;<BASE>;<CAP>;<SFFFF> *% ANGSTROM SIGN **(the letter Å really)***

<U01FB> <S00E5>;"<BASE><AIGUT>";"<MIN><MIN>";"<SFFFF><SFFFF>"

% LATIN SMALL LETTER A WITH RING ABOVE AND ACUTE

<U01FA> <S00E5>;"<BASE><AIGUT>";"<CAP><MIN>";"<SFFFF><SFFFF>"

% LATIN CAPITAL LETTER A WITH RING ABOVE AND ACUTE

*% The letter **Ä**:*

<U00E4> <S00E4>;<BASE>;<MIN>;<SFFFF> *% LATIN SMALL LETTER A WITH DIAERESIS*

<U0061_0308> <S00E4>;<BASE>;<MIN>;<SFFFF> *% decomposition of ä*

<U00C4> <S00E4>;<BASE>;<CAP>;<SFFFF> *% LATIN CAPITAL LETTER A WITH DIAERESIS*

<U0041_0308> <S00E4>;<BASE>;<CAP>;<SFFFF> *% decomposition of Ä*

<U01DF> <S00E4>;"<BASE><MACRO>";"<MIN><MIN>";"<SFFFF><SFFFF>"

% LATIN SMALL LETTER A WITH DIAERESIS AND MACRON

<U01DE> <S00E4>;"<BASE><MACRO>";"<CAP><MIN>";"<SFFFF><SFFFF>"

% LATIN CAPITAL LETTER A WITH DIAERESIS AND MACRON

reorder-end

6.5 Name of the Common Template Table and name declaration

The name ISO14651_2020_TABLE1 shall be used whenever the Common Template Table is referred to externally as a base point in a given context, whether in a process, contract, or procurement requirement. If another name is used due to practical constraints, a declaration of conformance shall indicate the correspondence between this other name and the name ISO14651_2020_TABLE1.

The use of a defined name is necessary to manage the different stages of development of this table. This follows from the nature of the reference character repertoire, for which development will be ongoing for a number of years or even decades.

Annex A
(normative)

Common Template Table

In order to minimize formatting problems and the risk of errors in reproduction, the Common Template Table is provided separately in a machine-readable file as a normative component of this document. The file name for this language version is different from the normative reference name specified in 6.5 due to the existence of file versions commented in other natural languages. The file for this language version can also be retrieved at the following URL.

— https://standards.iso.org/iso-iec/14651/ed-6/en/

There is an official French version of the file which only differs in its comments (its technical content is identical), and its name is: ISO14651_2020_TABLE1_fr.txt

NOTE 1 This document deprecates, but does not preclude specific reference to, the previous tables, which contained and still contain ordering information applicable to the repertoires of previous versions of ISO/IEC 10646 and their amendments. The previous tables can be found at the following URLs.

— [ordering information on the repertoire of characters as defined in ISO/IEC 10646-1:1993 including Amendments 1-9] https://standards.iso.org/iso-iec/14651/ed-1/ISO14651_2000_TABLE1_en.txt

— [ordering information on the combined repertoire of characters of ISO/IEC 10646-1:2000 and ISO/IEC 10646-2:2001] https://standards.iso.org/iso-iec/14651/ed-1/ISO14651_2002_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in ISO/IEC 10646:2003] https://standards.iso.org/iso-iec/14651/ed-1/ISO14651_2003_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in ISO/IEC 10646:2003 including Amendments 1-2] https://standards.iso.org/iso-iec/14651/ed-2/ISO14651_2006_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in ISO/IEC 10646:2003 including Amendments 1-4] https://standards.iso.org/iso-iec/14651/ed-2/ISO14651_2008_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in ISO/IEC 10646:2011] https://standards.iso.org/iso-iec/14651/ed-3/ISO14651_2010_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in ISO/IEC 10646:2012] https://standards.iso.org/iso-iec/14651/ed-3/ISO14651_2012_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in ISO/IEC 10646:2017] https://standards.iso.org/iso-iec/14651/ed-5/en/ISO14651_2017_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in the Unicode 7.0 standard] https://standards.iso.org/iso-iec/14651/ed-4/ISO14651_2014_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in the Unicode 8.0] https://standards.iso.org/iso-iec/14651/ed-4/ISO14651_2015_TABLE1_en.txt

— [ordering information on the repertoire of characters as defined in the Unicode 9.0] https://standards.iso.org/iso-iec/14651/ed-4/ISO14651_2016_TABLE1_en.txt

The current Common Template Table reflects the repertoire of characters as defined in ISO/IEC 10646:2020.

NOTE 2 The repertoire targeted by this document is equivalent to the repertoire of *The Unicode Standard Version 13.0, published by The Unicode Consortium*.

To meet cultural requirements of specific communities, delta declarations will have to be applied to the amended table as defined in this document.

ISO_14651_2020_TABLE1 is the name that is used for referring to this table in this version of this document.

Annex B
(informative)

Example tailoring deltas

B.1 Example 1 — Minimal tailoring

The following is a minimal tailoring of the Common Template Table.

```
reorder_after <SFFFF>
order_start forward;forward;forward;forward
reorder-end
```

B.2 Example 2 — Reversing the order of lowercase and uppercase letters

The following is a simple tailoring example to show how to reverse the order of uppercase versus lowercase from the order specified in the Common Template Table.

```
% Make uppercase letters sort before lowercase
% and scanning of accents done forward at level 2.

% The entire range of tertiary weight symbols
% <MIN>..<CIRCLE> are moved after <CIRCLECAP>, so that they order after
% <CAP> <WIDECAP> <COMPATCAP <FONTCAP> <CIRCLECAP> in the same
% relative order with respect to themselves. This has the effect of
% also making all the compatibility uppercase letters sort before
% their respective compatibility lowercase letters. (For example,
% U24B6 CIRCLED LATIN CAPITAL LETTER A will sort before
% U24D0 CIRCLED LATIN SMALL LETTER A.

% To do this correctly, an order_start is
% inserted to make the delta conformant.

reorder-after <CIRCLECAP>
<MIN>
<WIDE>
<COMPAT>
<FONT>
<CIRCLE>

reorder_after <SFFFF>
order_start forward;forward;forward;forward,position
reorder-end

% End of the uppercase/lowercase tailoring
```

B.3 Example 3 — Canadian delta and benchmark

This annex describes benchmark 1, based on Canadian standard CAN/CSA Z243.4.1-1998 (and 1992). The delta that precedes the benchmark *has been simplified* for illustration here; a larger delta is required, mainly for special characters, for full conformance to this Canadian standard, and is given here as an example only, limited to what is required for the benchmark. For complete information, the Canadian standard CAN/CSA Z243.4.1 should be consulted. The example tailoring is to be applied to the Common Template Table of Annex A, with the following delta.

a) Level processing properties:

forward; backward; forward; forward,position

b) Number of levels: Four (unchanged).

c) No symbol changes.

d) The following ordering changes are done:

— æ sorted as if it were separate letters "ae" at level 1. The letters "ae" are distinguished at level 2 from the character "æ" and is ordered before it;

— ð sorted as if it were the letter "d" at level 1. The letter "ð" is distinguished at level 2 from the letter "d" and is ordered after it;

— þ sorted as if it were separate letters "th" at level 1. The letters "th" is distinguished at level 2 from the letter "þ" and is ordered before it.

A Canadian tailoring expressed in the tailoring syntax for this document (normative only for Annex A) can be:

```
% copy ISO14651_2002_TABLE1
reorder-after <SFFFF>
order_start forward;backward;forward;forward,position

<U00E6> "<S0061><S0065>";"<BASE><VRNT1><BASE>";"<MIN><COMPAT><MIN>";
                                               "<SFFFF><SFFFF><SFFFF>" % æ
<U00C6> "<S0061><S0065>";"<BASE><VRNT1><BASE>";"<CAP><COMPAT><CAP>";
                                               "<SFFFF><SFFFF><SFFFF>" % Æ
<U01E3> "<S0061><S0065>";"<BASE><VRNT1><BASE><MACRO>";
  "<MIN><COMPAT><MIN><MIN>";"<SFFFF><SFFFF><SFFFF><SFFFF>" % æ WITH MACRON
<U01E2> "<S0061><S0065>";"<BASE><VRNT1><BASE><MACRO>";
  "<CAP><COMPAT><CAP><MIN>";"<SFFFF><SFFFF><SFFFF><SFFFF>" % Æ WITH MACRON
<U01FD> "<S0061><S0065>";"<BASE><VRNT1><BASE><AIGUT>";
              "<MIN><COMPAT><MIN><MIN>";"<SFFFF><SFFFF><SFFFF><SFFFF>" % ǽ
<U01FC> "<S0061><S0065>";"<BASE><VRNT1><BASE><AIGUT>";
              "<CAP><COMPAT><CAP><MIN>";"<SFFFF><SFFFF><SFFFF><SFFFF>" % Ǽ
<U00F0> <S0064>;<VRNT1>;<MIN>;<SFFFF> % ð
<U00D0> <S0064>;<VRNT1>;<CAP>;<SFFFF> % Ð
<U00FE> "<S0074><S0068>";"<BASE><VRNT1><BASE>";"<MIN><COMPAT><MIN>";
                                               "<SFFFF><SFFFF><SFFFF>" % þ
<U00DE> "<S0074><S0068>";"<BASE><VRNT1><BASE>";"<CAP><COMPAT><CAP>";
                                               "<SFFFF><SFFFF><SFFFF>" % Þ

reorder-end
```

Unordered list (required test case as per Canadian standard CAN/CSA Z243.4.1-1998, plus additions)			
ou	pêcher	Grossist	août
lésé	les	vice-presidents' offices	NOËL
péché	CÔTÉ	Copenhagen	@@@@@
vice-président	résumé	côte	L'Haÿ-les-Roses
9999	Ålborg	McArthur	CÔTE
OÙ	cañon	Mc Mahon	COTE
haïe	du	Aalborg	côté
coop	haie	Größe	coté
caennais	pécher	vice-president's offices	aide
lèse	Mc Arthur	cølibat	air
dû	cote	PÉCHÉ	vice-president
air@@@	colon	COOP	modelé
côlon	l'âme	@@@air	Thorvardur
bohème	resume	VICE-VERSA	MODÈLE
gêné	élève	gêne	maçon
meðal	Þorvarður	CO-OP	MÂCON

Unordered list (required test case as per Canadian standard CAN/CSA Z243.4.1-1998, plus additions)			
lamé	Canon	révélé	pèche
pêche	lame	révèle	pêché
LÈS	Bohême	çà et là	medal
vice versa	0000	MacArthur	ovoïde
C.A.F.	relève	Noël	pechère
Þorsmörk	gène	île	ode
cæsium	casanier	aïeul	péchère
resumé	élevé	Île d'Orléans	œil
Bohémien	COTÉ	nôtre	
co-op	relevé	notre	

List with required results as per Canadian standard CAN/CSA Z243.4.1-1998			
@@@@@	COOP	lamé	pêche
0000	CO-OP	les	péché
9999	Copenhagen	LÈS	PÉCHÉ
Aalborg	cote	lèse	pêché
aide	COTE	lésé	pécher
aïeul	côte	L'Haÿ-les-Roses	pêcher
air	CÔTE	MacArthur	pechère
@@@air	coté	MÂCON	péchère
air@@@	COTÉ	maçon	relève
Ålborg	côté	McArthur	relevé
août	CÔTÉ	Mc Arthur	resume
bohème	du	Mc Mahon	resumé
Bohême	dû	medal	résumé
Bohémien	élève	meðal	révèle
caennais	élevé	MODÈLE	révélé
cæsium	gène	Modelé	Þorsmörk
çà et là	gêne	Noël	Thorvardur
C.A.F.	gêné	NOËL	Þorvarður
Canon	Größe	notre	vice-president
cañon	Grossist	nôtre	vice-président
casanier	haie	ode	vice-president's offices
cølibat	haïe	œil	vice-presidents' offices
colon	île	ou	vice versa
côlon	Île d'Orléans	OÙ	VICE-VERSA
coop	lame	ovoïde	
co-op	l'âme	pèche	

B.4 Example 4 — Danish delta and benchmark

The following is a Danish example tailoring delta. This formal specification corresponds to Danish standard DS 377 and to "Retskrivningsordbogen", the Danish orthography specification.

```
% This tailoring is in accordance with Danish Standard DS 377 (1980)
% and the Danish Orthography Dictionary (Retskrivningsordbogen par 4,
% 1986). It is also in accordance with Greenlandic orthography.
```

```
collating-symbol <LIGHT> % Symbolic weight for lighter than <BASE>

% Define collating elements for <AA> - LETTER A WITH RING ABOVE
% and combinations with combining accents
collating-symbol <A-A> % symbolic weight for <AA>
collating-element <A-plus-combining-ring> from "<U0041><U030A>"
collating-element <a-plus-combining-ring> from "<U0061><U030A>"

% Define collating elements for sequences of a-plus-a
collating-element <A-plus-A> from "<U0041><U0041>"
collating-element <A-plus-a> from "<U0041><U0061>"
collating-element <a-plus-A> from "<U0061><U0041>"
collating-element <a-plus-a> from "<U0061><U0061>"

% Define collating elements for combinations with combining accents
collating-element <U-plus-combining-diaeresis> from "<U0055><U0308>"
collating-element <u-plus-combining-diaeresis> from "<U0075><U0308>"
collating-element <U-plus-combining-doubleacute> from "<U0055><U030B>"
collating-element <u-plus-combining-doubleacute> from "<U0075><U030B>"
collating-element <O-plus-combining-diaeresis> from "<U004F><U0308>"
collating-element <o-plus-combining-diaeresis> from "<U006F><U0308>"
collating-element <O-plus-combining-doubleacute> from "<U004F><U030B>"
collating-element <o-plus-combining-doubleacute> from "<U006F><U030B>"
collating-element <A-plus-combining-diaeresis> from "<U0041><U0308>"
collating-element <a-plus-combining-diaeresis> from "<U0061><U0308>"

% Add the obligatory order_start line.
reorder-after <SFFFF>
order_start forward;backward;forward;forward,position

% copy ISO14651_2002_TABLE1

% Make capital letters sort before lowercase.
% Cf. example 2 for more explanation.
reorder-after <CIRCLECAP>
<CAP>
<WIDECAP>
<COMPATCAP>
<FONTCAP>
<CIRCLECAP>
<MIN>
<WIDE>
<COMPAT>
<FONT>
<CIRCLE>

% Introduce a weight that is lighter than <BASE>
reorder-after <BASE>
<LIGHT>
<BASE>

% A list of reweighting statements to deal with specific collation
% behaviour for Danish. All of these define or redefine weight_list's,
% and so the entire block could simply be reordered after the
% order-start entry in the table. However, for clarity here and for
% stability, each separate set of weightings is reordered locally in
% the table around the first entry for that set of weightings.

% Actually a number of other reweighting statements should be specified
% with respect to the ISO/IEC 14651 table so that all accents be
% ignored on the first level, while the 14651 table distiguish
% for example between different accented versions of <l> and a number
% of other latin letters. This is considered less important
% and too elaborate for this example.

% Also this example delta does not include specifications for reordering
% of special characters like the currency denominators DOLLAR SIGN,
% CENT SIGN, and POUND SIGN, and unit denominators like the
% ANGSTROM SIGN, which have first-level weights in the CTT, while
% rules in DS 377 prescribe that special characters are ignored.
% Reorder Danish letters at the end of the alphabet, after z

reorder-after <S007A> % z
<S00E6> % <AE> - LETTER AE
<S00F8> % <O/> - LETTER O WITH STROKE
<A-A>   % <AA> - LETTER A WITH RING ABOVE

reorder-after <U007A> %  z - this *line* only given for stability
% The letter ae is a separate letter in Danish

<U00C6> <S00E6>;<BASE>;<CAP>;<SFFFF> % AE
<U00E6> <S00E6>;<BASE>;<MIN>;<SFFFF> % ae
<U01FC> <S00E6>;"<BASE><AIGUT>";"<CAP><MIN>";"<SFFFF><SFFFF>"
                                                          % AE WITH ACUTE
<U01FD> <S00E6>;"<BASE><AIGUT>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                          % ae WITH ACUTE

% The letter <a:> is given the same primary
% weight as <ae>, with unique variant weights at the secondary level

<U00C4> <S00E6>;"<BASE><VRNT1>";"<CAP><MIN>";"<SFFFF><SFFFF>"
                                                       % A WITH DIAERESIS
<U00E4> <S00E6>;"<BASE><VRNT1>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                      % a WITH DIAERESIS

% And replicate the weighting for the collating-element's formed with
combining accents

<A-plus-combining-diaeresis> <S00E6>;"<BASE><VRNT1>";"<CAP><MIN>";
                                                       "<SFFFF><SFFFF>"
<a-plus-combining-diaeresis> <S00E6>;"<BASE><VRNT1>";"<MIN><MIN>";
                                                       "<SFFFF><SFFFF>"

% The letter <o/> - O WITH STROKE - is a separate letter in Danish

<U00D8> <S00F8>;<BASE>;<CAP>;<SFFFF> % <O/>
<U00F8> <S00F8>;<BASE>;<MIN>;<SFFFF> % <o/>
<U01FE> <S00F8>;"<BASE><AIGUT>";"<CAP><MIN>";"<SFFFF><SFFFF>"
                                                      % <O/> WITH ACUTE
<U01FF> <S00F8>;"<BASE><AIGUT>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                      % <o/> WITH ACUTE

% The letters <o:> and <o"> are given the same primary
% weight as <o/>, with unique variant weights at the secondary level

<U00D6> <S00F8>;"<BASE><VRNT1>";"<CAP><MIN>";"<SFFFF><SFFFF>"
                                                       % O WITH DIAERESIS
<U00F6> <S00F8>;"<BASE><VRNT1>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                       % o WITH DIAERESIS
<U0150> <S00F8>;"<BASE><VRNT2>";"<CAP><MIN>";"<SFFFF><SFFFF>"
                                                    % O WITH DOUBLE ACUTE
<U0151> <S00F8>;"<BASE><VRNT2>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                    % o WITH DOUBLE ACUTE

% Replicate the weighting for the collating-element's formed with
                                                        combining accents

<O-plus-combining-diaeresis> <S00F8>;"<BASE><VRNT1>";"<CAP><MIN>";
                                                        "<SFFFF><SFFFF>"
<o-plus-combining-diaeresis> <S00F8>;"<BASE><VRNT1>";"<MIN><MIN>";
                                                        "<SFFFF><SFFFF>"
<O-plus-combining-doubleacute> <S00F8>;"<BASE><VRNT2>";"<CAP><MIN>";
                                                        "<SFFFF><SFFFF>"
<o-plus-combining-doubleacute> <S00F8>;"<BASE><VRNT2>";"<MIN><MIN>";
                                                        "<SFFFF><SFFFF>"

% The letter <aa> - A WITH RING ABOVE - is weighted following the
letter
                                                        <o/> (see above)
```

```
<U00C5> <A-A>;<BASE>;<CAP>;<SFFFF> % <AA>
<U00E5> <A-A>;<BASE>;<MIN>;<SFFFF> % <aa>
<U01FA> <A-A>;"<BASE><AIGUT>";"<CAP><MIN>";"<SFFFF><SFFFF>"
                                                       % <AA> WITH ACUTE
<U01FB> <A-A>;"<BASE><AIGUT>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                       % <aa> WITH ACUTE

% And replicate the weighting for the collating-element's formed with
                                                      combining accents

<A-plus-combining-ring> <A-A>;<BASE>;<CAP>;<SFFFF>
<a-plus-combining-ring> <A-A>;<BASE>;<MIN>;<SFFFF>

% The sequences of letters a-plus-a are weighted as secondary variants of
                                                                  <AA>

<A-plus-A> <A-A>;"<BASE><VRNT1>";"<CAP><CAP>";"<SFFFF><SFFFF>" % AA
<A-plus-a> <A-A>;"<BASE><VRNT1>";"<CAP><MIN>";"<SFFFF><SFFFF>" % Aa
<a-plus-A> <A-A>;"<BASE><VRNT1>";"<MIN><CAP>";"<SFFFF><SFFFF>" % aA
<a-plus-a> <A-A>;"<BASE><VRNT1>";"<MIN><MIN>";"<SFFFF><SFFFF>" % aa

% The letters u with diaeresis and u with double-acute are given the same
% primary weight as y, with unique variant weights at the secondary level

reorder-after <U00DC> % this *line* only given for stability
<U00DC> <S0079>;"<BASE><VRNT1>";"<CAP><MIN>";"<SFFFF><SFFFF>"
                                                       % U WITH DIAERESIS
<U00FC> <S0079>;"<BASE><VRNT1>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                      % u WITH DIAERESIS
<U0170> <S0079>;"<BASE><VRNT2>";"<CAP><MIN>";"<SFFFF><SFFFF>"
                                                    % U WITH DOUBLE ACUTE
<U0171> <S0079>;"<BASE><VRNT2>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                    % u WITH DOUBLE ACUTE

% And replicate the weighting for the collating-element's formed with
                                                    combining accents

<U-plus-combining-diaeresis> <S0079>;"<BASE><VRNT1>";"<CAP><MIN>";
                                                        "<SFFFF><SFFFF>"
<u-plus-combining-diaeresis> <S0079>;"<BASE><VRNT1>";"<MIN><MIN>";
                                                        "<SFFFF><SFFFF>"
<U-plus-combining-doubleacute> <S0079>;"<BASE><VRNT2>";"<CAP><MIN>";
                                                        "<SFFFF><SFFFF>"
<u-plus-combining-doubleacute> <S0079>;"<BASE><VRNT2>";"<MIN><MIN>";
                                                        "<SFFFF><SFFFF>"

% The letter eth is equated to d
% with a secondary difference to distinguish it from d

reorder-after <U0064> % this *line* only given for stability
<U00D0> <S0064>;"<BASE><VRNT1>";"<CAP><MIN>";"<SFFFF><SFFFF>" % ETH
<U00F0> <S0064>;"<BASE><VRNT1>";"<MIN><MIN>";"<SFFFF><SFFFF>" % eth

% The letter thorn is treated as a sequence of t + h, with a variant
% weight at the secondary level

reorder-after <U00DE> % this *line* only given for stability
<U00DE> "<S0074><S0068>";"<BASE><VRNT1><BASE>";"<CAP><MIN><MIN>";
                                          "<SFFFF><SFFFF><SFFFF>" % THORN
<U00FE> "<S0074><S0068>";"<BASE><VRNT1><BASE>";"<MIN><MIN><MIN>";
                                          "<SFFFF><SFFFF><SFFFF>" % thorn

% The letter oe is treated as a sequence of o + e, with a special weight
% at the secondary level

reorder-after <U006F> % this *line* only given for stability
<U0152> "<S006F><S0065>";"<BASE><LIGHT>";"<CAP><MIN>";
                                                   "<SFFFF><SFFFF>" % OE
<U0153> "<S006F><S0065>";"<BASE><LIGHT>";"<MIN><MIN>";
                                                   "<SFFFF><SFFFF>" % oe
```

```
% Space, hyphen-minus, hyphen, and solidus are given a primary weight
% before any letter or digit, with hyphen-minus and solidus
% given a secondary difference from the weight for space.

reorder-after <U0020> % this *line* only given for stability
<U0020> <S0020>;<BASE>;<MIN>;<U0020> % SPACE
<U002D> <S0020>;"<BASE><VRNT1>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                        % HYPHEN-MINUS
<U2010> <S0020>;"<BASE><VRNT1>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                              % HYPHEN
<U002F> <S0020>;"<BASE><VRNT2>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                              % SOLIDUS

% The letter kra (for Greenlandic) is equated to a lowercase q,
% with a secondary difference to distinguish it from q itself

reorder-after <U0071> % q - this *line* only given for stability
<U0138> <S0071>;"<BASE><VRNT1>";"<MIN><MIN>";"<SFFFF><SFFFF>" % kra

% The letter sharp s is treated as a sequence of s + s
% with a special weight at the secondary level to make it come
% before s-plus-s - shorter precedes longer.

reorder-after <U0073> % s - this *line* only given for stability
<U00DF> "<S0073><S0073>;"<BASE><LIGHT>";"<MIN><MIN>";"<SFFFF><SFFFF>"
                                                             % SHARP S

% To facilitate deterministic ordering, all controls have a unique
% weight at the 4th level.

reorder-after <U0000>
<U0000>..<U001F> IGNORE;IGNORE;IGNORE;<U0000>..<U001F>
<U007F>..<U009F> IGNORE;IGNORE;IGNORE;<U007F>..<U009F>

reorder-end

% End of the example tailoring for Danish
```

Benchmark for Danish (sorted order)			
A/S	D.S.B.	RÉE, A	STORM PETERSEN
ANDRE	DSC	REE, B	STORMLY
ANDRÉ	EKSTRA-ARBEJDE	RÉE, L	THORVALD
ANDREAS	EKSTRABUD	REE, V	THORVARDUR
AS	EKSTRAARBEJDE	SCHYTT, B	ÞORVARÐUR
CA	HØST	SCHYTT, H	THYGESEN
ÇA	HAAG	SCHÜTT, H	VESTERGÅRD, A
CB	HÅNDBOG	SCHYTT, L	VESTERGAARD, A
ÇC	HAANDVÆRKSBANKEN	SCHÜTT, M	VESTERGÅRD, B
DA	Karl	ß	ÆBLE
ÐA	karl	SS	ÄBLE
DB	NIELS JØRGEN	SSA	ØBERG
ÐC	NIELS-JØRGEN	STORE VILDMOSE	ÖBERG
DSB	NIELSEN	STOREKÆR	Århus

B.5 Example 5 — A tailoring for Khmer

The Khmer script is mainly used in Cambodia. The tailoring given below is not included in the CTT (see Annex A) itself in order to keep the CTT simple, especially for rare letterforms. For example, the Khmer ROBAT for which the tailoring below may not be desirable for efficiency reasons, since this letter occurs very rarely, but the tailoring for handling it correctly may affect the efficiency of collation also for texts that do not contain any ROBAT.

```
reorder-after <MAX>

% Khmer:
collating-symbol <S1794_S17C9> % KHMER LETTER BA, KHMER SIGN MUUSIKATOAN
collating-symbol <S1794_S17CA> % KHMER LETTER BA, KHMER SIGN TRIISAP
collating-symbol <S17BB_S17C6> % KHMER VOWEL SIGN U, KHMER SIGN NIKAHIT
collating-symbol <S17B6_S17C6> % KHMER VOWEL SIGN AA, KHMER SIGN NIKAHIT
collating-symbol <C1780>..<C179C>

% Declaration of Khmer contractions
collating-element <U1794_17C9> from "<U1794><U17C9>" % KHMER LETTER BA,
                                                 KHMER SIGN MUUSIKATOAN

collating-element <U1794_17CA> from "<U1794><U17CA>"
                                   % KHMER LETTER BA, KHMER SIGN TRIISAP
collating-element <SW_17CC_1780>..<SW_17CC_17A2> from
                             "<U1780>..<U17A2><U17CC>" % KHMER LETTER KA,
                      KHMER SIGN ROBAT..KHMER LETTER QA, KHMER SIGN ROBAT
collating-element <SW_17CC_17A5>..<SW_17CC_17B3> from
                  "<U17A5>..<U17B3><U17CC>" % KHMER INDEPENDENT VOWEL QI,
KHMER SIGN ROBAT..KHMER INDEPENDENT VOWEL QAU, KHMER SIGN ROBAT
collating-element <U17C6_17BB> from "<U17BB><U17C6>"
                                 % KHMER VOWEL SIGN U, KHMER SIGN NIKAHIT
                                                    (OM properly spelled)
collating-element <U17BB_17C6> from "<U17C6><U17BB>"
                                 % KHMER SIGN NIKAHIT, KHMER VOWEL SIGN U
                           (OM with the wrong sequence of the characters)
collating-element <U17C6_17B6> from "<U17B6><U17C6>"
                                % KHMER VOWEL SIGN AA, KHMER SIGN NIKAHIT
                                                    (AM properly spelled)
collating-element <U17B6_17C6> from "<U17C6><U17B6>"
                                % KHMER SIGN NIKAHIT, KHMER VOWEL SIGN AA
                           (AM with the wrong sequence of the characters)
collating-element <U17D2_1780>..<U17D2_179C> from
               "<U17D2><U1780>..<U179C>" % COENG, KHMER LETTER KA..COENG,
                                                         KHMER LETTER QA
collating-element <U17D2_17A5>..<U17D2_17B3> from
                                      "<U17D2><U17A5>..<U17B3>" % COENG,
        KHMER INDEPENDENT VOWEL QI..COENG, KHMER INDEPENDENT VOWEL QAU

reorder-after <S1794> % KHMER LETTER BA
<S1794_17C9> % KHMER LETTER BA, KHMER SIGN MUUSIKATOAN
<S1794_17CA> % KHMER LETTER BA, KHMER SIGN TRIISAP

reorder-after <S17C5> KHMER VOWEL SIGN AU
<S17BB_17C6> % KHMER VOWEL SIGN U, KHMER SIGN NIKAHIT

reorder-after <S17C6> KHMER SIGN NIKAHIT
<S17B6_17C6> % KHMER VOWEL SIGN AA, KHMER SIGN NIKAHIT

reorder-after <S17D2>
<C1780>..<C1794> % COENG, KHMER LETTER KA..COENG, KHMER LETTER BA
<C1795>..<C179A> % COENG, KHMER LETTER PHA..COENG, KHMER LETTER RO
<C17AB> % COENG, KHMER INDEPENDENT VOWEL RY
<C17AC> % COENG, KHMER INDEPENDENT VOWEL RYY
<C179B> % COENG, KHMER LETTER LO
<C17AD> % COENG, KHMER INDEPENDENT VOWEL LY
<C17AE> % COENG, KHMER INDEPENDENT VOWEL LYY
<C179C>..<C17A2> % COENG, KHMER LETTER VO..COENG, KHMER LETTER QA

reorder-after <SFFFF>
order_start forward;forward;forward;forward

<U1794_17C9> <S1794_17C9>;<BASE>;<MIN>;<SFFFF> % KHMER LETTER BA,
                                                 KHMER SIGN MUUSIKATOAN
<U1794_17CA> <S1794_17CA>;<BASE>;<MIN>;<SFFFF> % KHMER LETTER BA,
                                                 KHMER SIGN TRIISAP

%% The ROBAT contractions should be used only in an "advanced" tailoring
%% for Khmer, since ROBAT is rather rarely used, and these contractions
%% may impact on the efficiency of the key computation even if ROBAT does
%% not occur, since these contractions begin with commonly used letters.

<SW_17CC_1780>..<SW_17CC_17A2> "<S179A><S17D2><S1780>..<S17A2>";
   "<BASE><VRNT1><BASE><BASE>";
   "<MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF>"
   % KHMER LETTER KA, KHMER SIGN ROBAT..KHMER LETTER QA, KHMER SIGN ROBAT

<SW_17CC_17A5>..<SW_17CC_17A6> "<S179A><S17D2><S17A2><S17B7>..<S17B8>";
   "<BASE><VRNT1><BASE><BASE><VRNT1><BASE>";
   "<MIN><MIN><MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF><SFFFF><SFFFF><SFFFF>"
   % KHMER INDEPENDENT VOWEL QI,
      KHMER SIGN ROBAT..KHMER INDEPENDENT VOWEL QII, KHMER SIGN ROBAT

<SW_17CC_17A7> "<S179A><S17D2><S17A2><S17BB>";
   "<BASE><VRNT1><BASE><BASE><VRNT1><BASE>";
   "<MIN><MIN><MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF><SFFFF><SFFFF><SFFFF>"
   % KHMER INDEPENDENT VOWEL QU, KHMER SIGN ROBAT

<SW_17CC_17A8> "<S179A><S17D2><S17A2><S17BB>";
   "<BASE><VRNT1><BASE><BASE><VRNT2><BASE>";
   "<MIN><MIN><MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF><SFFFF><SFFFF><SFFFF>"
   % KHMER INDEPENDENT VOWEL QUK, KHMER SIGN ROBAT

<SW_17CC_17A9> "<S179A><S17D2><S17A2><S17BC>";
   "<BASE><VRNT1><BASE><BASE><VRNT1><BASE>";
   "<MIN><MIN><MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF><SFFFF><SFFFF><SFFFF>"
   % KHMER INDEPENDENT VOWEL QUU, KHMER SIGN ROBAT

<SW_17CC_17AA> "<S179A><S17D2><S17A2><S17BC>";
   "<BASE><VRNT1><BASE><BASE><VRNT2><BASE>";
   "<MIN><MIN><MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF><SFFFF><SFFFF><SFFFF>"
   % KHMER INDEPENDENT VOWEL QUUV, KHMER SIGN ROBAT

<SW_17CC_17AF>..<SW_17CC_17B1> "<S179A><S17D2><S17A2><S17C2>..<S17C4>";
   "<BASE><VRNT1><BASE><BASE><VRNT1><BASE>";
   "<MIN><MIN><MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF><SFFFF><SFFFF><SFFFF>"
   % KHMER INDEPENDENT VOWEL QE,
      KHMER SIGN ROBAT..KHMER INDEPENDENT VOWEL QOO TYPE ONE,
      KHMER SIGN ROBAT

<SW_17CC_17B2> "<S179A><S17D2><S17A2><S17C4>";
   "<BASE><VRNT1><BASE><BASE><VRNT2><BASE>";
   "<MIN><MIN><MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF><SFFFF><SFFFF><SFFFF>"
   % KHMER INDEPENDENT VOWEL QOO TYPE TWO;  KHMER SIGN ROBAT

<SW_17CC_17B3> "<S179A><S17D2><S17A2><S17C5>";
   "<BASE><VRNT1><BASE><BASE><VRNT1><BASE>";
   "<MIN><MIN><MIN><MIN><MIN><MIN>";
   "<SFFFF><SFFFF><SFFFF><SFFFF><SFFFF><SFFFF>"
   % KHMER INDEPENDENT VOWEL QAU, KHMER SIGN ROBAT

%%% Khmer OM and AAM (the NIKAHIT should be written after the vowel):
<U17BB_17C6> <S17BB_17C6>;<BASE>;<MIN>;<SFFFF>
                                % KHMER VOWEL SIGN U, KHMER SIGN NIKAHIT
<U17C6_17BB> <S17BB_17C6>;<BASE>;<MIN>;<SFFFF>
                                % KHMER SIGN NIKAHIT, KHMER VOWEL SIGN U
<U17B6_17C6> <S17B6_17C6>;<BASE>;<MIN>;<SFFFF>
                               % KHMER VOWEL SIGN AA, KHMER SIGN NIKAHIT
<U17C6_17B6> <S17B6_17C6>;<BASE>;<MIN>;<SFFFF>
                               % KHMER SIGN NIKAHIT, KHMER VOWEL SIGN AA

reorder-end
```

Annex C
(informative)

Preparation

C.1 General considerations

Preparation is necessary only for modification and/or duplication of original strings to render them context-independent prior to the comparison phase. A non-duplicating preparation maps a given string to one string. A non-duplicating preparation can be composed with the key generation and comparison, as, for example, is needed for Lao and Thai (see C.2), or proper ordering of numerals (see C.3). A duplicating preparation can map a given string to several strings (to be sorted).

Examples of non-duplicating preparations are the following:

- vowel-consonant rearrangement, as is needed for Thai (see C.2) and Lao;
- transformation of numbers so that the result will be ordered in numerical order, as opposed to positional order (see C.3). Numeric ordering is particularly delicate and requires special consideration in many cases;
- removal or rotation of characters that are a nuisance for special requirements of ordering; for example, removing articles (language dependent) in sorting book names as in

```
Tale of two cities, A
```

- transformation of abbreviated data into a fuller form. For example: transformation of "McArthur" to give "MacArthur".

Some examples of a duplicating preparation are the following:

- duplicating a string into several “rotations”, like when producing a keyword-in-context index;

	International string ordering
International string	ordering
International	string ordering

- duplication of a string such as "41" for as it is spelled out in different languages (Irish Gaelic, German, English, and French).

```
daichead a haon
einundvierzig
forty-one
quarante et un
```

C.2 Thai string ordering

C.2.1 General

This Clause explains some of the principles behind the tailoring of the CTT given in B.5, as well as the CTT ordering for Thai (and to some extent Lao).

C.2.2 Thai ordering principles

The widely accepted standard for Thai lexicographical ordering is defined in the Royal Institute Dictionary 2542 B.E. Edition (1999 A.D.), the official standard Thai dictionary. The ordering principles, expanded to cover the full range of the Thai script and its computerisation are as follows.

— Words are ordered alphabetically, not phonetically. Consonants order is:

ก ข ฃ ค ฅ ฆ ง จ ฉ ช ซ ฌ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท

ธ น บ ป ผ ฝ พ ฟ ภ ม ย ร ฤ ล ฦ ว ศ ษ ส ห ฬ อ ฮ

— Vowels, and nikhahit, are also ordered by written forms, not by sounds. Vowels and nikhahit order is:

-ํ -ะ -ั -า -ๅ -ำ -ิ -ี -ึ -ื -ุ -ู เ- แ- โ- ใ- ไ- -ฺ

อ ว ย are always ordered as consonants, although they sometimes act as vowels;

ๅ is a long-legged variant of า, used with the long-legged consonants ฤ and ฦ: ฤๅ and ฦๅ;

ำ is logically an า followed by a ํ. However, the Unicode compatibility decomposition of the precomposed character is to a ํ followed by an า, so this misspelling shall be handled as well.

— No syllable structure or word boundary analysis is required, as Thai lexicons are ordered alphabetically, not phonetically. Note that Thai normally does not use any word separator, except, and exceptionally, zero width space.

— String comparison is performed from left to right, but treating leading vowels (เ- แ- โ- ใ- ไ-, corresponding to characters U0E40-U0E44), as though they followed the consonant that immediately follows them. Therefore, rearrangement (in some way) is needed before comparison.

— Tones and diacritics are ignored at level 1. At level 2 their order is:

-๎ -์ -็ -่ -้ -๊ -๋

— Since Thai, Lao, and Khmer are uncased, it may seem that the third level is not needed for Thai, Lao, and Khmer string ordering. However, the third level is used to differentiate LAKKHANGYAO (ๅ) as a variant of SARA AA (า), since similar variants in other scripts are differentiated in a like manner at level 3, as well as for other variation cases in Thai, Lao, and Khmer.

— When Thai punctuation marks (๛ ๚ ฯ ๏ ๆ) are concerned, another level of weights is required for them. This corresponds to the fourth level in the Common Template Table. In string ordering, punctuation marks are less significant than any tone marks and diacritics, and shall be ignored in all the first three levels. Note that PAIYANNOI (ฯ) and THAI CHARACTER MAIYAMOK (repeat mark ๆ) are regarded and ordered as punctuation marks, not letters, despite their Unicode general category as "Lo" and "Lm" respectively. For example, "ข้างๆ, ข้างกบ, ข้างๆ คูๆ, ข้างจัน" is a valid order in the Royal Institute Dictionary. In the first level, the considered weights correspond to ขาง, ขางกบ, ขางคู, and ขางจัน respectively.

— The ten Thai decimal digits (๐ ๑ ๒ ๓ ๔ ๕ ๖ ๗ ๘ ๙), each semantically equivalent to Arabic digit 0-9, respectively. Their weights are then equal to their corresponding Arabic digit in the first level, and are different in the second level, to distinguish script.

C.2.3 Vowel/consonant rearrangement

Regarding the handling of pre-vowels, either a collation preparation or collating-element grouping (as in the tailoring in B.5) is required. The collation preparation scans the string once and swaps every leading Thai vowel with its succeeding character (ideally only if the succeeding character is a Thai consonant). The prepared string is then passed to the normal weight calculation process. Another way to manage this is by means of collating-element formation — the approach taken both by the CTT of this document and by the collation weighting table of the Unicode Collation Algorithm.[27] Every possible pair of leading vowel and consonant is defined as a collating-element, whose weight is equal to that of the rearranged substring. In addition, since two เ in sequence look just like a แ, two เ in sequence should be handled just like a แ.

Note that the rearrangement of each leading vowel is simply performed with its immediate succeeding consonant. No consonant cluster analysis is needed. Indeed, doing so would result in ambiguities or yield a different order than that specified in the Royal Institute Dictionary. Below are the examples.

a) Ambiguities: The problem with ambiguity is illustrated by the word “เพลา”. It has two potential pronunciations: either as a two-syllable word, “phe-la” (meaning “time”), or as a one-syllable word, “phlao” (meaning “axle” or “abate”). A rearrangement algorithm which follows the distinct pronunciation of the potential cluster ‘พล’ in this string would result in distinct keys, “พเลา” and “พลเา”, and therefore different weights, which are equally legal. Both words need to have the same weight to be sortable, however.

b) Non-conforming ordering: To illustrate the difference in ordering caused by the treatment of consonant clusters, consider these words, shown in conforming order: “เพล, เพลง, เพศ”. The correct rearrangement ignores any clusters and results in the following: “พเล, พเลง, พเศ”, which sorts in the order shown. If, however, pairs of consonants that form legal clusters were grouped as single collation elements (regardless of actual pronunciation where the potential pronunciation is ambiguous), then the results of rearrangement would be “<พล>เ, <พล>เง, พเศ”, which would yield the (non-conforming) ordering “เพศ, เพล, เพลง”. Again, if actual clusters were grouped as single collation elements (with some disambiguation effort), then the results of rearrangement would be “พเล, <พล>เง, พเศ”, which would yield the (non-conforming) ordering “เพล, เพศ, เพลง”.

C.2.4 Example ordered strings

Here is an ordering example: Example for Thai (sorted order)			
กก	โกน	แข่งขัน	ผัด
กรรม	โกร๋น	แขน	ฯพณฯ
กรรม์	ใกล้	ครรภ-	พณิชย์
-กระแย่ง	ไก่	ครรภ์	ย่อง
กราบ	ไกล	จุมพล	รอง
กะเกณฑ์	ข้น	จุํพล	ฤทธิ์
กัก	ขนาบ	ชาย	ฤษี
ก้าว	ข้าง	เฒ่า	ฤๅษี
กำ	ข้างๆ	เณร	ลลิตา
กิน	ข้างกระดาน	ตลาด	ภาชา
กี่	ข้างขึ้น	ทูลเกล้า	วก
กึ๋น	ข้างควาย	ทูลเกล้าฯ	ศาล
กุน	ข้างๆ คูๆ	ทูลเกล้าทูลกระหม่อม	หริภุญชัย
กูด	ข้างเงิน	น้า	หฤทัย
เก้ง	ข้างออก	น้ำ	หลง
เกล้า	เข็ด	นี้	แหง่
เกลียว	เขน	บุญหลง	แห่ง
เก้า	เข็น	บุญ-หลง	แหนม
เกาะ	เข่น	ปา	แหนหวง

Here is an ordering example: Example for Thai (sorted order)			
เกี่ยว	แข็ง	ป่า	แหบ
เกี้ยะ	แข่ง	ป้า	แหม
เกือก	แข้ง	ป๊า	อาน
แกง	แข้งขวา	ป๋า	ฮา
แกะ	แข็งขัน	ปาน	

C.3 Handling of numeral substrings in collation

C.3.1 General

A numeral is a string representing a number. The examples here deal with numerals that represent values in *R*, the real numbers, or really, subsets of *R*, as these have a predetermined order. Only decimal numerals are dealt with in the examples given here.

The presentation below will first give positional system decimal numerals for natural numbers using the digits 0-9. It will progress to numerals for whole numbers, numerals with a fraction part, a fraction part, and an exponent. There is also a brief discussion on numerals with digits from other scripts, scripts that sometimes uses another syntax with digits for numerals (such as Hàn numerals), and Roman numerals. There are circumstances where digits do not represent numerical values, such as in part numbers and telephone numbers. The preparations described below have undesirable consequences in cases where *apparent* numerals do not represent numerical values, such as when the ordering telephone numbers or part “numbers”, and should be avoided in those cases.

C.3.2 Handling of ”ordinary” numerals for natural numbers

The Common Template Table has no means of ordering strings with numbers in such a way that the resulting order reflects the number values represented by the numerals. For example, given the following randomly-arranged strings:

— Release 1

— Release 20

— Release 12

— Release 2

— Release 9

the method described in this document yields the following order for these strings:

— Release 1

— Release 12

— Release 2

— Release 20

— Release 9

(It is sufficient simply to look positionally at just the first digit in each numeral to see why this ordering results.) A more acceptable ordering is as follows.

— Release 1

- Release 2
- Release 9
- Release 12
- Release 20

The Common Template Table defined in this document cannot be tailored to give this result. However, preparation can be done prior to the basic collation step to achieve the desired results when numeric value order is desired. The prepared strings are normally not presented to the user; only the original strings are. The prepared strings are normally only used for the collation key construction. A simple, but not very general, way of preparing numerals for natural ordering is to pad them with zeroes to a given number of digits. If one pads the numerals in our original example strings up to three digits, the result will be the following.

- Release 001
- Release 020
- Release 012
- Release 002
- Release 009

Using the Common Template Table defined in this document, one then obtains the strings in a better order (here showing the strings as they are after preparation, which are normally not shown in the result).

- Release 001
- Release 002
- Release 009
- Release 012
- Release 020

However, there are two problems with this approach.

- One needs to determine beforehand a (usually small) number of digits to pad up to. If the number of digits to pad up to is too large, the strings after preparation can become rather long, especially if there are several numerals in each string. If the number of digits to pad up to is too small, however, the risk is greater that there are actually occurring numerals with more digits than one has padded up to, which results in partially getting back to the original situation, where the numerals' values are not taken entirely into account.
- Determinacy is lost, if some of the original numerals were already partially zero-padded. For example, if the original strings were:
  - Release 01
  - Release 1

  the strings after preparation are identical, and the end result (as the user would normally see it) could be either:
  - Release 01
  - Release 1

  or

- Release 1
- Release 01

and the relative order may come out differently for different occurrences of numerals or different runs of the collation process applying the same rules. This kind of indeterminacy is undesirable.

There are many ways to deal with these problems. The following is one such way.

To each maximal digit subsequence prepend a fixed-number-of-digits numeral which represents the original number of digits in the numeral. For most cases, a two-digit count would suffice (allowing up to 99 digits in the original integer numerals). For example, given the original strings:

- Release 1
- Release 01
- Release 20
- Release 12
- Release 2
- Release 09
- Release 9

one obtains after this preparation the following strings:

- Release 011
- Release 0201
- Release 0220
- Release 0212
- Release 012
- Release 0209
- Release 019

which would be ordered by the basic mechanism of this document to:

- Release 011
- Release 012
- Release 019
- Release 0201
- Release 0209
- Release 0212
- Release 0220

and are normally presented to the user as:

- Release 1
- Release 2

- Release 9
- Release 01
- Release 09
- Release 12
- Release 20

This particular method puts numerals with a like original number of digits close to each other, even if the actual value represented is smaller due to the original zero-padding. If the represented *values* should be kept close together, one should instead duplicate the numeral: first, a count of digits for the leading-zero-stripped numeral, the leading-zero-stripped numeral itself, followed by the original numeral. The duplication is needed to get determinacy relative to the original strings. For example, using the same original strings as above:

- Release 011 1
- Release 011 01
- Release 0220 20
- Release 0212 12
- Release 012 2
- Release 019 09
- Release 019 9

which would be ordered by the basic mechanism of this document to:

- Release 011 01
- Release 011 1
- Release 012 2
- Release 019 09
- Release 019 9
- Release 0212 12
- Release 0220 20

and normally be presented to the user as:

- Release 01
- Release 1
- Release 2
- Release 09
- Release 9
- Release 12
- Release 20

The originally zero-padded numerals consistently come before the numeral without (or with less) original zero-padding. The preparation processing could move the original numerals (in order of occurrence) to the very end of each string, if one wants to give the original zero-padding lesser significance than the text following the numerals.

The presence of several natural numerals in each string causes no additional problem.

Taking care of the natural number numerals is, in most cases sufficient, and it is recommended that it be included as part of the usual preparation of strings to be collated. However, such preparation is not required by this document.

C.3.3 Handling of positional numerals in other scripts

ISO/IEC 10646 encodes decimal digits for a number of scripts. In most cases, these are used in a positional system, just like 0-9 usually are. However, one should not regard a sequence of numerals mixed from different scripts as a single numeral; rather, one should consider each maximal substring of digits of the same script to be a numeral.

C.3.4 Handling of other non-pure positional system numerals or non-positional system numerals (e.g. Roman numerals)

Chinese and some other languages can use decimal digits (in the Hàn script, for instance) interspersed with ideographs for “one thousand”, “ten”, etc. If such numerals are to be collated according to the value they represent, one can proceed as above, adding a step just after the initial duplication: convert the copy to the corresponding positional system numeral in the syntax used here for whole numerals.

Roman numerals, if handled, can be handled in a similar fashion to that described above. Duplicate and replace the first copy with the same natural number expressed in the decimal positional system. For example, “Louis V”, where the V is determined to be a Roman numeral, can be modified to “Louis 5 V”.

Caveat: In this case, human interactive intervention or an expert system may be required, as in the following example involving the French language: CHAPITRE DIX might mean CHAPTER 10 or CHAPTER 509 ("dix" is the French word for 10, it is also the Roman numeral for 509).

C.3.5 Handling of numerals for whole numbers

If numerals for negative whole numbers are also to be ordered according to their value, there are a number of issues to be considered. Most frequently, negative whole values are given numerals that begin with a negation sign. The negation sign may be HYPHEN-MINUS U002D (but this character may often represent a true hyphen, rather than a negation), or MINUS SIGN U2212. However, there are other conventions also, like using a SOLIDUS U002F or a PERCENT SIGN U0025 to indicate negativity; or the negation indicator can come *after* the digits rather than before; or negativity can be indicated by putting the digits between parenthesis, and/or putting the digits in a contrasting colour (often red). In the examples here, only the case that negativity is indicated by an immediately prepended MINUS SIGN is dealt with. Positivity is indicated by either the absence of a MINUS SIGN, or the presence of a PLUS SIGN U002B.

Example strings:

— Temperature: −9 °C

— Temperature: 0 °C

— Temperature: −14 °C

— Temperature: 05 °C

— Temperature: +5 °C

— Temperature: −0 °C

— Temperature: −09 °C

- Temperature: 105 °C
- Temperature: +05 °C
- Temperature: 5 °C

One preparation to get an acceptable and determinate order for numerals (in this syntax) for whole numbers is as follows (actual implementations should do something equivalent, but more efficient).

a) Duplicate the numerals in the string (including sign indications), putting the "original" ones (not to be touched by the following steps) in order of original occurrence at the end or the string, leaving the copies to be modified at the original positions. This step ensures determinacy.

b) Ensure that all of the copies have an explicit initial sign indicator.

c) Remove leading zeroes in the copies of the numerals (systematically either leaving one zero digit for zero or representing 0 by the empty string of digits); alternatively, let all numeral copies have exactly one leading zero.

d) Between the sign indicator and the digits in the copies of the numerals, insert a (two-digit) count of how many digits there were (after removing the leading zeroes).

e) Do 9s complement on each digit in each copy of a negated numeral. 9s complement of a digit that individually represents the value *x*, is 9–*x*. That is, 9s complement of 0 is 9, of 9 is 0, of 5 is 4, etc.

For the basic collation step, use a tailoring of the template given in this document, namely, a tailoring where the PLUS SIGN and the MINUS SIGN are significant at the same level as the digits, and where the MINUS SIGN has less weight than the PLUS SIGN (in the example below, it is assumed that the weight of PLUS SIGN is less than the weight of 0, but this is not a prerequisite for getting an acceptable ordering).

Our example strings after this preparation:

- Temperature: –980 °C –9
- Temperature: +00 °C 0
- Temperature: –9 785 °C –14
- Temperature: +015 °C 05
- Temperature: +015 °C +5
- Temperature: –99 °C –0
- Temperature: –980 °C –09
- Temperature: +03 105 °C 105
- Temperature: +015 °C +05
- Temperature: +015 °C 5

The ordering for these, using the basic mechanism of this document, is:

- Temperature: –9 785 °C –14
- Temperature: –980 °C –09
- Temperature: –980 °C –9
- Temperature: –99 °C –0
- Temperature: +00 °C 0

- Temperature: +015 °C +05
- Temperature: +015 °C +5
- Temperature: +015 °C 05
- Temperature: +015 °C 5
- Temperature: +03 105 °C 105

and normally presented to the user as:

- Temperature: –14 °C
- Temperature: –09 °C
- Temperature: –9 °C
- Temperature: –0 °C
- Temperature: 0 °C
- Temperature: +05 °C
- Temperature: +5 °C
- Temperature: 05 °C
- Temperature: 5 °C
- Temperature: 105 °C

This preparation results in a determinate ordering of strings that have zero or more numerals for whole numbers in them, such that the numerals are ordered according to the integer value they represent.

The process for other syntaxes for whole numbers can be similar. Just add a step to convert the copies to the syntax used here for whole numbers.

This technique for handling negative numerals can be used also for numerals with a fractional part and so on (see C.3.6).

C.3.6 Handling of positive positional numerals with fractional parts

The method presented in C.3.5 can easily be adapted to the case where fraction parts may occur and are to be taken into account. A problem is, however, that the characters often used to delimit the integer part from the fraction part are also used for other purposes. he separator character is generally either FULL STOP U002E, or COMMA U002C. These characters also have other uses, also in conjunction with digits.

For the examples here, assume that FULL STOP is used (only) as a fraction part delimiter.

Do as in C.3.5, but count only the digits in the integer part of the numeral for the count of digits to be prepended. The fraction part delimiter character (here: FULL STOP) can be removed.

For example:

- –12.34
- 12.34
- 3.1415
- 3.14

After preparation:

- –978765 –12.34
- +021234 12.34
- +013.1415 3.1415
- +01314 3.14

Ordered:

- –978765 –12.34
- +01314 3.14
- +0131415 3.1415
- +021234 12.34

As presented to the user:

- –12.34
- 3.14
- 3.1415
- 12.34

C.3.7 Handling of positive positional numerals with fraction parts and exponent parts

For very large, or very small, values, one often uses formats like $2{,}5 \times 10^{7}$ (to illustrate just one possible way of writing these for the purposes of these examples). Here, there is already an exponent, which shall be combined with the “number of integer part digits” (here: digits before the decimal point), by adding those two numbers to get a resulting fixed-number-of-digits exponent to prepend just before the first digit. For this example, with a three-digit exponent: we get +00825. One problem here is that the resulting exponent may be negative. To handle this, use an exponent bias. For a three-digit exponent a bias of 500 may be suitable, which gives us for this example numeral: +50825, and for the numeral $2{,}5 \times 10^{-7}$, we get +49425. Negative values are handled as before, with 9’s complement. $-2{,}5 \times 10^{7}$ gives –49174, and $-2{,}5 \times 10^{-7}$ gives –50574. This method should be familiar to anyone with knowledge about (radix 10) floating point arithmetic.

Thus:

- $2{,}5 \times 10^{-7}$
- $-2{,}5 \times 10^{7}$
- $2{,}5 \times 10^{7}$
- $-2{,}5 \times 10^{-7}$

After preparation (including a duplicate of the original, for determinacy):

- +49425 $2{,}5 \times 10^{-7}$
- –49174 $-2{,}5 \times 10^{7}$
- +50825 $2{,}5 \times 10^{7}$
- –50574 $-2{,}5 \times 10^{-7}$

Ordered:

- −49174 −2,5 × 10^{7}
- −50574 −2,5 × 10^{-7}
- +49425 2,5 × 10^{-7}
- +50825 2,5 × 10^{7}

As presented to the user:

- −2,5 × 10^{7}
- −2,5 × 10^{-7}
- 2,5 × 10^{-7}
- 2,5 × 10^{7}

C.3.8 Handling of date and time of day indications

Going a bit beyond plain numerals, date and time-of-day indications often employ numerals (as well as names for months, weekdays, etc.) for the parts of the date and time-of-day indication. It is not uncommon to want to order this kind of information also when it occurs within strings.

The preparation needed to obtain date and time-of-day indications, of some predetermined syntaxes, ordered according to point in time is similar to what has been described in C.3 in general:

a) Duplicate all date and time-of-day indications to maintain determinacy of collation when the original strings differ, but point in time identical. Leave the originals at the end of the strings, untouched by the following steps.

b) Convert the copies of the date and time indications to the same calendar system, if there are several calendar (sub)systems used and handled. The calendar (sub)system converted to, shall be suitable for being able to get proper time order. We will here use the Gregorian calendar system and the subsystem of year, month, day-of-month.

c) Put the date and time-of-day elements in order of decreasing significance (to the resolution taken into account): Full year, month, day-of-month, hour, minute, second, fraction of second.

d) Use a 24-hour/day clock for the time-of-day indications. Remove A.M. or P.M. indications, if present and handled, in the date-time indication copies.

e) Use the UTC (Co-ordinated Universal Time) time zone for the date and time-of-day indications. Remove time zone indications, if present, in the date-time indication copies.

f) Use month numbers, rather than month names. Use two digits each for month, day-of-month, hour, minute, second.

g) Use full year number representation, as many digits as needed. Take abbreviations into account so that the full year number is used. For example, ‘98’ might denote year 98 or year 1998, or 1898, etc. No indeterminacy regarding year due to abbreviations like these may be present after the preparation step.

h) For years AD, use an initial PLUS SIGN. For years BC, use an initial MINUS SIGN. Remove the original AD or BC indication from the copies. (To nitpick, year *n* BC should be represented by year (1−*n*), which is less or equal to zero if *n* is positive.)

i) For the year indications, insert between the sign indication and the first digit for the year indication a digit telling how many digits there are in the full year indication. One digit for this should suffice.

j) For negative years, replace each digit in the year indication (including the digit telling the number of digits in the original full year indication) with its 9s complement digit.

k) Make sure the textual format for all of the date indication copies is the same (paying attention to hyphens, spaces, etc.). (This is most easily accomplished by printing them in the same format from an internal, non-string, representation.)

l) Alternatively, use a number indicating the point of time on a linear time scale (for example, hours, milliseconds, or days from a predetermined point in time), to the resolution desired, and handle this as an ordinary numeral (see above).

For the basic collation step, use a tailoring of the template given in this document. Use a tailoring where the PLUS SIGN and the MINUS SIGN are significant at the same level as the digits, and where the MINUS SIGN has less weight than the PLUS SIGN.

For example:

— Dated: July 19, 1955, at 1 p.m. GMT

— Dated: January, 20 BC

— Dated: Sept. 20, 1995, at 1 p.m. PST

— Dated: 11-june/345 AD

After preparation:

— Dated: +41955-07-19T13:00Z July 19, 1955, at 1 p.m. GMT

— Dated: –780-01 January, 20 BC

— Dated: +1995-09-20T10:00Z Sept. 20, 1995, at 1 p.m. PST

— Dated: +3345-06-11 11-june/345 AD

Ordered:

— Dated: –780-01 January, 20 BC

— Dated: +3345-06-11 11-june/345 AD

— Dated: +41955-07-19T13:00Z July 19, 1955, at 1 p.m. GMT

— Dated: +41995-09-20T10:00Z Sept. 20, 1995, at 1 p.m. PST

As presented to the user:

— Dated: January, 20 BC

— Dated: 11-june/345 AD

— Dated: July 19, 1955, at 1 p.m. GMT

— Dated: Sept. 20, 1995, at 1 p.m. PST

C.3.9 Making numbers less significant than letters

In many cases, numerals preceding letters should be considered as less significant than the following alphabetic part. However, the Common Template Table specifies digits to be level 1 significant. To make numerals less significant than letters, either tailor the weight table so that numerals are ignored at level 1 (but significant at level 2 or 3), or alternatively leave them significant at level 1, but prepare the strings so that numerals are moved to the end of the string or moved to a less significant field. When doing such a move, one needs to pay attention not to map different strings to identical strings (or identical string fields), so that determinacy is maintained (see C.3.8).

Some examples where it is appropriate to consider numerals as less significant than letters: street or block names with one or more numbers to indicate where in the street/block, if that/those number(s) precede the street or block name (common for example in the US and in Japan); chemical compound names which have prepended numerals, for example, 1,2-diclorobenzol.

C.3.10 Maintaining determinacy

As noted in C.3.8 in several cases, part of the string has been duplicated to maintain determinacy in collation, when the original strings are different, but when preparation may otherwise turn different strings into identical strings.

This method of concatenating a copy of a substring in order to maintain determinacy can be used more generally, so if there are several preparations affecting different parts of the strings, one may simply duplicate the original strings to begin with, and only perform the preparation (without additional copying) on the given string, then concatenating on the initial copy.

One disadvantage with just concatenating the two copies is that the base letters of the second half of the “doubled” string count as more significant than the accents and case of the resulting first half of the “doubled” string. This document has no mechanism for handling this in a better way, where the “original” (the second half of the “doubled” string) would count as less significant than the entire first half of the “doubled” string. This may be handled better by having the original and copy in different “fields”, and construct the collation key by combining the full keys for each “field”. Such processing is beyond the scope of this document, however.

Maintenance of determinacy when some of the original strings to be collated are identical, is out of the scope of this document. A sorting processor should document if it is “stable” (maintaining initial relative order of identical strings) or not. This is useful to know when sorting on one field of multi-field data.

C.4 Pre-processing Hangul

C.4.1 General

In general, one needs to handle not only modern complete Hangul syllables, but also old and/or incomplete syllables to collate Hangul data properly. When Hangul data in ISO/IEC 10646 is input to an algorithm supporting this document, it is not always collated properly. Therefore, one needs to pre-process these data so that this problem can be fixed. Of course, depending on the specific collation desired (for example, the Republic of Korea and DPRK have different collation orders), some tailoring of the CTT table may be required.

The main purpose of the pre-processing is, after syllable boundaries are determined, to transform one complete or incomplete syllable into 9 simple letters.

Suppose that one has Hangul data represented using Hangul letters (Jamos) U11xx/UA9xx/D7xx and/or Hangul Wanseong syllables UAC00..UD7A3. The pre-processing steps 1 to 3 are as described here.

C.4.2 Step 1

If a syllable is represented with a Wanseong syllable character UAC00.. UD7A3, it is decomposed into syllable-initial, syllable-peak and optionally syllable-final parts as follows.

Find the syllable component indices I, P, F calculated in ISO/IEC 10646:2011, 24.7, item 4).

a) The code position of a syllable-initial letter is U+1100 + I. Example: If I is 17 (0x11 in hexadecimal), then the code position is U+1111.

b) The code position of a syllable-peak letter is U+1161 + P. Example: If P is 16 (0x10 in hexadecimal), then the code position is U+1171.

c) If F is zero, there is no syllable-final letter. Otherwise, the code position of a syllable-final letter is U+11A7 + F. Example: If F is 18 (0x12 in hexadecimal), then the code position is U+11B9.