ISO/IEC JTC1/SC 22/WG 20 N 731

Date: 2000-04-06

ISO
ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ

CEI (IEC)
COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE
INTERNATIONAL ELECTROTECHNICAL COMMISSION
МЕЖДУНАРОДНАЯ ЗЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ

Title:	ISO/IEC DIS 14651 – International string ordering and comparison – Method for comparing character strings and description of the common template tailorable ordering *[ISO/CEI DIS 14651 – Classement international de chaînes de caractères – Méthode de comparaison de chaînes de caractères et description du modèle commun et adaptable de classement]*
Status:	Final Draft International Standard
Reference:	SC22/WG20 N 670 (Disposition of comments on FCD 14651.2)
Date:	2000-04-06
Project:	22.30.02.02
Editor:	Alain LaBonté Gouvernement du Québec Secrétariat du Conseil du trésor 875, Grande Allée Est, Section 3C Québec, QC G1R 5R8 Canada
Email:	alb@sct.gouv.qc.ca

[end of cover letter]

[title page: to be provided by ITTF]

ISO/IEC DIS 14651

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

Classement international de chaînes de caractères – Méthode de comparaison de chaînes de caractères et description du modèle commun et adaptable de classement

[copyright notice, library info, …]

Contents

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.

In the field of information technology, ISO and IEC have established a joint technical committee known as ISO/IEC JTC1. Draft International Standards adopted by the joint technical committee are circulated to the national bodies for voting. Publication as an international standard requires approval by at least 75% of the national bodies that cast a vote.

The ISO/IEC 14651 International Standard has been prepared by the Joint Technical Committee ISO/IEC JTC1, Information Technology.

Introduction

This International Standard 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 International Standard requires that all deviations from the Template, called "deltas", be declared to document resultant discrepancies.

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

The Draft Technical Report ISO/IEC DTR 14652 (under development) has specifications for ordering that informatively complements the specifications in this International Standard, and where additional information may be sought on ordering keywords defined in this International Standard.

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

1. Scope

This International Standard defines:

- A reference comparison method. This method is applicable to two character strings to determine their relative order. The method can be applied to strings containing characters from the full repertoire of ISO/IEC 10646-1. 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, standardised 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 International Standard or from one of its tailorings.
- 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 International Standard.
- 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 first edition of ISO/IEC 10646-1 up to Amendment 7. 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 UCS.

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

NOTE 2: The character repertoire used in this International Standard is equivalent to that of the Unicode Standard version 2.1.

- 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 multiple-octet coded character set.
- Requirements for a declaration of the differences (delta) between the collation table and the Common Template Table.

This International Standard does *not* mandate:

- 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 International standard (see informative Annex C).

NOTE 2: Although no user interface is required to choose options or to specify tailoring of the Common Template Table, conformance requires always declaring the applicable delta, a declaration of differences with this table. It is recommended that processes present available tailoring options to users.

2. Conformance

A process is conformant to this International Standard if it meets the requirements prescribed in subclauses 6.2 to 6.5.

A declaration of conformity to this International Standard 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 clause 6.4 and how many levels are defined in the delta.

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.

3. Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

Members of IEC and ISO maintain registers of currently valid International Standards.

- ISO/IEC 10646-1:1993 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*.
- ISO/IEC 10646-1:1993/Amd.1:1996 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane Amendment 1: Transformation Format for 16 planes of group 00 (UTF-16)*.
- ISO/IEC 10646-1:1993/Amd.2:1996 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane Amendment 2: UCS Transformation Format 8 (UTF-8)*.
- ISO/IEC 10646-1:1993/Amd.4:1996 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane Amendment 4*.
- ISO/IEC 10646-1:1993/Amd.5:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane Amendment 5: Hangul syllables*.
- ISO/IEC 10646-1:1993/Amd.6:1997 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane Amendment 6: Tibetan*.

- ISO/IEC 10646-1:1993/Amd.7:1997 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane Amendment 7: 33 additional characters*.
- ISO/IEC 10646-1:1993/Amd.9:1997 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane Amendment 9: Identifiers for characters*.
- ISO/IEC 10646-1:1993/Amd.18:1997 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane Amendment 18: Symbols and other characters*.

NOTE: Only the EURO SIGN and the OBJECT REPLACEMENT CHARACTER from Amendment 18 are accounted for in Annex A of this International Standard at this time.

4. Definitions

For the purposes of this International Standard, the following definitions apply:

4.1 character string a sequence of characters

4.2 collation equivalent to the term “ordering”

4.3 collating symbol a symbol used to specify weights assigned to a collating element

4.4 collation (weighting) table
a mapping from collating elements to weighting elements

4.5 collating element a sequence of one or more characters that are considered a single element for ordering

4.6 delta differences from a given collation table. The given collation table, together with a given delta, forms a new collation table. Unless otherwise specified in this International Standard, the term "delta" always refers to differences from the Common Template Table as defined in this International Standard

4.7 (collation) level the sequence number for a subkey

4.8 ordering a process by which two strings are determined to be in exactly one of the relationships of **less than**, **greater than**, or **equal** to one another

4.9 ordering key a sequence of subkeys used to determine an order

4.10 (collation) preparation
a process in which given character strings are mapped to (other) character strings logically before the calculation of the ordering key for each of the strings

4.11 reference comparison method
the method for establishing an order between two ordering keys (see clause 6)

4.12 subkey a sequence of weights computed for a character string for a particular level.

4.13 symbol collating element

4.14 (collation) weight a positive integer value, used in subkeys, reflecting the relative order of collating elements

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

5. Symbols and abbreviations

Following ISO/IEC 10646-1:1993, Amendment 9, characters are referenced as U*XXXX* where *X* stands for any hexadecimal digit (using upper case letters where applicable) and refers to the value of that character in ISO/IEC 10646-1. This convention is used throughout this International Standard.

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

6. String comparison

6.1. Preparation of character strings prior to comparison

It may be necessary to transform character strings before the reference comparison method is applied to them (see annex C for an example of such preparation). Although not part of the scope of this International Standard, preparation may be an important part of the ordering process. See Annex C for some examples of preparation.

Where applicable, 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 must 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.

NOTE 1: 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 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 may be invisible to the end-user, it is recommended that strings be normalized according to Unicode normalization to achieve this equivalence – see Bibliography, Unicode Technical Report no. 15.

NOTE 2: 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 3: 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 not below 3 levels, in tailoring.

NOTE: In the Common Template Table, levels generally have the following characteristics, although the purpose of each level 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 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 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 may 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.

An optional property of the last level of comparison, if that level is greater than three, is that before comparing weights of each “ignorable” character, a comparison on the numeric position of each such character in the two strings may be effected. This processing property is known as the ‘position’ option. In other words, for two strings equivalent at all levels except the last one, the string having an ignorable in the lowest position comes before the other one. In case corresponding “ignorables” 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.

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 lowest positions in the logical sequence of characters correspond to the rightmost characters of a string (from the point of view of their natural presentation sequence). Conversely, for the Latin script, written left to right, the lowest positions in the logical sequence of characters correspond to the leftmost characters of the string (from the point of view of 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 10646-1, the Arabic script is artificially separated into two pseudo-scripts: 1) the logical, intrinsic Arabic, coded independently of 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 10646-1 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 prehandling be used to make sure that Arabic presentation forms and other Arabic characters be fed to the comparison method in logical order.

A tailored table may be separated into sections for ease of tailoring. Each section is then assigned a name consistent with the specification in subclause 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

When two strings are to be compared to determine their relative order, the two strings are first broken up into a sequence of collating elements taking into accountthe multi-character "`collating-element`" statements declared and used in a tailored table (*if* the syntax of clause 6.3.1 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.

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.

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`".

If there is no entry in the tailored table for a character of the input string, then the character's weights are undefined. Characters with undefined weights should be ordered, with respect to characters that have defined weights, as if the undefined ones were given the weight named "UNDEFINED" at the first level. If there is no weight assignment to the symbol "UNDEFINED" before the symbol <SFFFF>'s weight assignment in a given tailored table, then the table shall be interpreted as if "UNDEFINED" was weighted just before <SFFFF> (the maximal level 1 weight). The ordering of characters with undefined weights with respect to other characters with undefined weights is not specified in this standard.

NOTE: A possible way to order characters with undefined weights is as if *there were tailoring lines like this one added to the table, in UCS code point order (call the maximal level 4 weight <PLAIN> here):*

<UXXXX> "<UNDEFINED><UXXXX>";<BASE>;<MIN>;<PLAIN>

6.2.2.1. Formation of a subkey with the "`forward`" level processing parameter

Subkeys, at a particular level, formed with the "`forward`" level processing parameter, are built in the following way:

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.

6.2.2.2. Formation of a subkey with the "`backward`" level processing parameter

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.

6.2.2.3. Formation of a subkey with the "`forward,position`" level processing parameter

Subkeys, at the last level, formed with the "`forward,position`" level processing parameter are formed by forming a subkey as with the "`forward`" parameter, but for collating elements that are not "`IGNORE`"d at all levels but the last one, their last level weighting (list of weights) is replaced by a single weight (call it <PLAIN> here) that is larger than all other weights at the last level in the given tailored table. Collating elements that are "`IGNORE`"d at all levels but the last one, retain their weighting according to the given tailored table. Finally, any trailing sequence of the maximal weight (<PLAIN>) is removed from the subkey, effectively replacing each trailing maximal weight with a zero weight.

NOTE: For any level, implementations are allowed to apply an order-preserving reduction of all subkeys at that level (see, e.g., Annex E). Such an order-preserving subkey reduction is useful for levels 2, 3, and 4. Level 2 often has long stretches of the weight named <BASE> in Annex A. Level 3 often has long stretches of the weight named <MIN> in Annex A. Level 4 often has long stretches of the weight named <PLAIN> here. One such ordering preserving subkey reduction technique effectively encodes, in the last level subkey, the (relative) position, as single number each, of each otherwise ignored character; hence the name of the "`position`" option.

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 subclause 6.2.2 of this International Standard:

- 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 below in this subclause. Keys can be compared either up to a given level, or up to the last level of the given tailored collation weighting table.

NOTE 1: The comparison may 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 may be found early in the keys. When a bigger set of strings are to be ordered, it may be advisable to generate the ordering keys, and store each key or an initial segment of each, before comparing the keys.*

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 2: 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:

- L_z is the length of the subkey z, i.e., the number of weights in the subkey.
- $z_{\mathbf{wt}(a)}$, where $1 \le a \le L_z$, is the weight at index position a (an integer > 0) of the subkey z.
- $u_{\mathbf{sk}(b)}$, where $1 \le b \le m$, 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 orders, defined for a given tailored collation table as follows:

1. Weights are positive integer values (in the reference method), and are compared as such for the purposes of collation.

2. A subkey v is *less than* a subkey w (written $v < w$) **if and only if** there exists an integer i, where $1 \le i \le L_v+1$ and $i \le L_w$, such that for all integers j, where $1 \le j < i$, the equality $v_{\mathbf{wt}(j)} = w_{\mathbf{wt}(j)}$ holds; and either $i \le L_v$ 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.

3. 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 $1 \le i \le s$ and $i \le m$, such that for all integers j, where $1 \le j < i$, 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.

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

NOTE 3: 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.

6.3. Common Template Table: formation and interpretation

This clause specifies:

- The syntax used to form the Common Template Table in Annex A of this International Standard 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.

6.3.1. BNF syntax rules for the Common Template Table in Annex A

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'  four_digit_hex_string  '>') |
                 ('<U-' eight_digit_hex_string '>') ;

simple_symbol =  '<' identifier '>' ;

collating_element =
                 'collating-element' space+ symbol space+
                 'from' space+ quoted_symbol_sequence ;

quoted_symbol_sequence =
                 '"' symbol+ '"' ;

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* ;

four_digit_hex_string =
                 hex_upper hex_upper hex_upper hex_upper ;

eight_digit_hex_string =
                 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> ;
```

6.3.1.1 Keyword usage

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

collating-symbol	Define a collating symbol representing a weight.
collating-element	Define a collating element symbol representing a multi-character collating element. This keyword is optional.
order_start	Define 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	Specify the end of the collation-order statements.
reorder-after	Redefine collating rules. Specify 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 elements's weightings.
reorder-end	Specify the end of the "reorder-after" collating order statements.
section	Define a section of the table. A section can be moved as a whole by reorder-section-after.
reorder-section-after	Redefine 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 elements's weightings.

6.3.2. Well-formedness conditions

WF 1. Any *simple_symbol* occurring in a *weight_list* shall also occur in the initial *symbol_element* of the *symbol_weight* in which that *weight_list* occurs, or in a *symbol_definition* that occurs earlier in the sequence of lines that constitute a *weight_table*.

NOTE: All simple_symbol*s must be "defined" before they are "used".*

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

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

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

NOTE: A tailorable table must be consistent in its use of levels throughout.

WF 4. A *tailored_table* shall contain one *order_start* statement. This 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 *direction*s as the number of *level_token*s of any *weight_list* in that *tailored_table*.

NOTE: Any order_start *must 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_token*s in that *weight_list* shall also consist of a *symbol_group*.

NOTE: IGNORE *must 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_line*s that constitutes a *tailored_table*.

NOTE: All section_identifier*s must be "defined" before they are "used".*

WF 8. No two *section_definition*s in a *tailored_table* shall contain the same values in their *section_identifier*s.

NOTE: Multiple definitions of section*s are prohibited;* section_identifier*s must 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. A *tailored_table* shall contain one *order_start* and one *order_end*.

WF 11. No *reorder_section_after* shall contain a *target_symbol* whose value is the same as any *symbol* in the *section_definition_list* whose *section_identifier* is the same as the *section_identifier* in that *reorder_section_after*.

NOTE: A section *must not be reordered after a line which the* section *itself contains; attempts at recursive relocation of lines are prohibited.*

WF 12. Any *symbol_range* shall contain two *symbol*s which meet the following conditions: 1) Each of the two *symbol*s shall contain a common prefix. 2) The portions of *identifier* of each of the two *symbol*s following the common prefix shall be a *hex_numeric_string.* 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 range 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 13. Any *symbol_weight* that contains more than one *symbol_range* shall contain only *symbol_range*s 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.*

6.3.3. Interpretation of tailored tables

I 1. A *section* consists either 1) of the list of *simple_line*s 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_line*s 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 entire table is encountered.*

I 2. A *simple_line* consisting of a *symbol_definition* containing a *symbol_range* is equivalent to a sequence of *simple_line*s, 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 must contain the same number of digits as the *hex_numeric_string* of the first symbol. The number of *simple_line*s 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_range*s is equivalent to a sequence of *simple_line*s, where each *symbol_range* has been expanded into a sequence of *symbol*s, as described in I 2 for *symbol_definition*s.

NOTE: A symbol_weight *of the form "*<U2000>..<U2002> <S0301>..<S0303>;<BLANK>;<MIN>; <U2000>..<U2002>*" is equivalent to the three lines:*
<U2000> <S0301>;<BLANK>;<MIN>;<U2000>
<U2001> <S0302>;<BLANK>;<MIN>;<U2001>
<U2002> <S0303>;<BLANK>;<MIN>;<U2002>

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

1. all *table_line*s that were ahead of the *reorder_after* and that contained *symbol_definition*s whose *symbol* matches the *symbol* of any *symbol_definition* in the *table_line*s between the *reorder_after* and *reorder_end* have been removed,

2. the *table_line*s between that *reorder_after* and the first subsequent *reorder_end* to immediately follow the first *table_line* in the *tailored_table* containing a *symbol_definition* whose *symbol* is the same as the *target_symbol* in the *reorder_after* have been reordered, and

3. that *reorder_after* and that *reorder_end* have been removed.

NOTE: 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_definition*s 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 may 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 *section-reorder_after* reordered (in the same relative order as the *table_line*s 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 or *reorder_section_after*s.

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

6.3.4. 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_assignment*s occur in the *weight_table*.

NOTE 1: The table_line*s 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_assignment*s are then mapped to a set of positive integers (weights) that varies monotonically with the set of line numbers.*

NOTE 2: This does not restrict the starting number for the weight of the first weight_assignment *(other than it must 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 collation-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 3: 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 4: In a tailored table the value of any hex_numeric_string *associated with a* symbol *typically does not reflect the numeric weighting of the* symbol.

6.3.5. 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.

NOTE: If one takes two strings, builds keys for each based on TBL1 *and compares them, one should always get the same results as when one builds keys for those strings based on* TBL2 *and compares them, if those two tables are claimed to be equivalent.*

6.3.6. Conditions for results to be considered equivalent

An implementation of international string ordering is conformant with this International Standard 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 clause 6.2.3 of this International Standard.

6.4. Declaration of a delta

Tailoring shall be based upon the Common Template Table described in annex A. Tailoring may be accomplished using any syntax that is equivalent to the one described in this International Standard.

NOTE 1: For example, ISO/IEC DTR 14652, uses a compatible extension of the syntax used in this International Standard for tailoring. A tailoring delta can also be expressed using the syntax of the Unicode collation algorithm (see Bibliography - Unicode Technical Report no. 10). 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 International Standard 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:

1. At least one valid *order_start* entry described in clause 6.3.1; an unlimited number of sections containing an *order_start* entry and an *order_end* entry may be declared.

2. The number of levels used for comparison.

3. The list of *symbol_definition* weights (as defined in 6.2.1) added and after which *symbol_definition* entry each insertion is made.

4. The list of *simple_line* entries (as defined in 6.2.1) deleted or inserted, referencing after which *simple_line* entry in the Common Template Table the insertions are made

NOTE 2: It is recommended that a delta should not be bigger than necessary.

In cases where a process has provision to allow the end-user to tailor the table himself or herself, a statement of conformance shall indicate which of the 4 elements of the previous list are tailorable and which of those 4 elements are not tailorable. For those which are not tailorable, the delta of fixed elements relative to the Common Template Table shall be declared.

NOTE 3: The declaration may 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 U+00E5 after U+00FE at the primary level.
Collate U+00E4 after U+00E5 at the primary level. "

or

"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. they have a weight value higher at level 1 than the one for thorn (þ), which itself comes after the ones for all variants of the letter z.

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

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

% Declare new collation symbols (weight names):
collation-symbol <S00E5> *% for å*
collation-symbol <S00E4> *% for ä*

% Declare new collating elements for the decompositions (substring names):
collation-element <U0061_030A> **from** "<U0061><U030A>" *% decomposition of å*
collation-element <U0041_030A> **from** "<U0041><U030A>" *% decomposition of Å*
collation-element <U0061_0308> **from** "<U0061><U0308>" *% decomposition of ä*
collation-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>;<U00E5> *% LATIN SMALL LETTER A WITH RING ABOVE*
<U0061_030A> <S00E5>;<BASE>;<MIN>;"<U0061_030A>" *% decomposition of å*
<U00C5> <S00E5>;<BASE>;<CAP>;<U00C5> *% LATIN CAPITAL LETTER A WITH RING ABOVE*
<U0041_030A> <S00E5>;<BASE>;<CAP>;"<U0041_030A>" *% decomposition of Å*
<U212B> <S00E5>;<BASE>;<CAP>;<U212B> *% ÅNGSTRÖM SIGN **(the letter Å really)***

<U01FB> <S00E5>;"<BASE><AIGUT>";"<MIN><MIN>";<U01FB> *% LATIN SMALL LETTER A WITH RING ABOVE AND ACUTE*
<U01FA> <S00E5>;"<BASE><AIGUT>";"<CAP><MIN>";<U01FA> *% LATIN CAPITAL LETTER A WITH RING ABOVE AND ACUTE*

% The letter **Ä***:*
<U00E4> <S00E4>;<BASE>;<MIN>;<U00E5> *% LATIN SMALL LETTER A WITH DIAERESIS*
<U0061_0308> <S00E4>;<BASE>;<MIN>;"<U0061_0308>" *% decomposition of ä*
<U00C4> <S00E4>;<BASE>;<CAP>;<U00C4> *% LATIN CAPITAL LETTER A WITH DIAERESIS*
<U0041_0308> <S00E4>;<BASE>;<CAP>;"<U0041_0308>" *% decomposition of Ä*

<U01DF> <S00E4>;"<BASE><MACRO>";"<MIN><MIN>";<U01DF> *% LATIN SMALL LETTER A WITH DIAERESIS AND MACRON*
<U01DE> <S00E4>;"<BASE><MACRO>";"<CAP><MIN>";<U01DE> *% LATIN CAPITAL LETTER A WITH DIAERESIS AND MACRON*
reorder-end

6.5. Name of the Common Template Table and name declaration

Whenever the Common Template Table is referred externally as a base point in a given context, whether in a process, contract, or procurement requirement, it shall be referenced using the name ISO14651_2000_TABLE1. If another name is used due to practical constraints, a declaration of

conformance shall indicate how the correspondence between this other name and the name ISO14651_2000_TABLE1 is taken care of.

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 – Common Template Table (normative)

In order to minimize formatting problems and the risk of bad reproduction, the common template table is provided separately in a machine-readable file as a normative component of this International standard under the name ISO14651_2000_TABLE1.txt and it can be retrieved on the ISO web site at the following URL: http://www.iso.ch/ittf/ISO14651_2000_TABLE1.htm

NOTE. the htm *link is redirected to the actual* txt *file on the ISO site.*

The current Common Template Table reflects the repertoire of characters as defined in ISO/IEC 10646-1:1993 including Amendments 1-9.

When ordering data applicable to ISO/IEC 10646-1:2000 becomes available, this International standard and specifically its Common Template Table will be amended accordingly to cover the ordering of the additional characters and scripts. To meet cultural requirements of specific communities, delta declarations will have to be applied to the amended table as defined in this International standard.

ISO14651_2000_TABLE1 is the name that is used for referring to this table in this version of this International standard.

Annex B – Example tailoring deltas (informative)

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,
% U+24B6 CIRCLED LATIN CAPITAL LETTER A will sort before
% U+24D0 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:

1. Level processing properties:

```
forward; backward; forward; forward,position
```

2. Number of levels: 4 (unchanged).

3. No symbol changes.

4. 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 International Standard (normative only for Annex A) can be:

```
% copy ISO14651_2000_TABLE1

reorder-after <SFFFF>
order_start forward;backward;forward;forward,position

<U00E6> "<S0061><S0065>";"<BASE><VRNT1><BASE>";"<MIN><COMPAT><MIN>";<U00E6>    % æ
<U00C6> "<S0061><S0065>";"<BASE><VRNT1><BASE>";"<CAP><COMPAT><CAP>";<U00C6>    % Æ

<U01E3> "<S0061><S0065>";"<BASE><VRNT1><BASE><MACRO>";"<MIN><COMPAT><MIN><MIN>";<U01E3>
         % æ WITH MACRON
<U01E2> "<S0061><S0065>";"<BASE><VRNT1><BASE><MACRO>";"<CAP><COMPAT><CAP><MIN>";<U01E2>
         % Æ WITH MACRON

<U01FD> "<S0061><S0065>";"<BASE><VRNT1><BASE><AIGUT>";"<MIN><COMPAT><MIN><MIN>";<U01FD> % ǽ
<U01FC> "<S0061><S0065>";"<BASE><VRNT1><BASE><AIGUT>";"<CAP><COMPAT><CAP><MIN>";<U01FC> % Ǽ


<U00F0> <S0064>;<VRNT1>;<MIN>;<U00F0>    % ð
<U00D0> <S0064>;<VRNT1>;<CAP>;<U00D0>    % Ð

<U00FE> "<S0074><S0068>";"<BASE><VRNT1><BASE>";"<MIN><COMPAT><MIN>";<U00FE>    % þ
<U00DE> "<S0074><S0068>";"<BASE><VRNT1><BASE>";"<CAP><COMPAT><CAP>";<U00DE>    % 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
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'Orléan	œil
Bohémien	COTÉ	nôtre	
co-op	relevé	notres	

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	medal	relevé
août	CÔTÉ	meðal	resume
bohème	du	McArthur	resumé
Bohême	dû	Mc Arthur	résumé
Bohémien	élève	Mc Mahon	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.

% copy ISO14651_2000_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>

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

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

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

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

% 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.


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

% The letter <a:> is given the same primary
% weight as <ae>, with unique variant weights at the secondary level.
<U00D6> <S00E6>;"<BASE><VRNT1>";"<CAP><MIN>";<U00D6> % A WITH DIAERESIS
<U00F6> <S00E6>;"<BASE><VRNT1>";"<MIN><MIN>";<U00F6> % a WITH DIAERESIS

% And replicate the weighting for the collating-element's formed with combining accents
<A+combining-diaeresis> <S00E6>;"<BASE><VRNT1>";"<CAP><MIN>";"<U0041><U0308>"
<a+combining-diaeresis> <S00E6>;"<BASE><VRNT1>";"<MIN><MIN>";"<U0061><U0308>"

% The letter <o/> - O WITH STROKE - is a separate letter in Danish
<U00D8> <S00F8>;<BASE>;<CAP>;<U00D8> % <O/>
<U00F8> <S00F8>;<BASE>;<MIN>;<U00F8> % <o/>
<U01FE> <S00F8>;"<BASE><AIGUT>";"<CAP><MIN>";<U01FE> % <O/> WITH ACUTE
<U01FF> <S00F8>;"<BASE><AIGUT>";"<MIN><MIN>";<U01FF> % <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>";<U00D6> % O WITH DIAERESIS
<U00F6> <S00F8>;"<BASE><VRNT1>";"<MIN><MIN>";<U00F6> % o WITH DIAERESIS
<U0150> <S00F8>;"<BASE><VRNT2>";"<CAP><MIN>";<U0150> % O WITH DOUBLE ACUTE
<U0151> <S00F8>;"<BASE><VRNT2>";"<MIN><MIN>";<U0151> % o WITH DOUBLE ACUTE

% Replicate the weighting for the collating-element's formed with combining accents
<O+combining-diaeresis> <S00F8>;"<BASE><VRNT1>";"<CAP><MIN>";"<U004F><U0308>"
<o+combining-diaeresis> <S00F8>;"<BASE><VRNT1>";"<MIN><MIN>";"<U006F><U0308>"
<O+combining-doubleacute> <S00F8>;"<BASE><VRNT2>";"<CAP><MIN>";"<U004F><U030B>"
<o+combining-doubleacute> <S00F8>;"<BASE><VRNT2>";"<MIN><MIN>";"<U006F><U030B>"

% The letter <aa> - A WITH RING ABOVE - is weighted following the letter <o/> (see above)
<U00C5> <A-A>;<BASE>;<CAP>;<U00C5> % <AA>
<U00E5> <A-A>;<BASE>;<CAP>;<U00E5> %  <aa>
%Although some consider that the Ångström sign is to be processed differently from the
%letter Å, some users will want to have this same glyph -- for which no difference exists
%on paper -- be treated as if it were the letter which it appears to be.
%If so, decomment the next line
%<U212B> <A-A>;<BASE>;<CAP>;<U212B> % ÅNGSTRÖM SIGN
<U01FA> <A-A>;"<BASE><AIGUT>";"<CAP><MIN>";<U01FA> % <AA> WITH ACUTE
<U01FB> <A-A>;"<BASE><AIGUT>";"<MIN><MIN>";<U01FB> % <aa> WITH ACUTE

% And replicate the weighting for the collating-element's formed with combining accents
<A+combining-ring> <A-A>;<BASE>;<CAP>;<U00C5>
<a+combining-ring> <A-A>;<BASE>;<MIN>;<U00E5>

% The sequences of letters a+a are weighted as secondary variants of <AA>
<A+A> <A-A>;"<BASE><VRNT1>";"<CAP><CAP>";"<U0041><U0041>" % AA
<A+a> <A-A>;"<BASE><VRNT1>";"<CAP><MIN>";"<U0041><U0061>" % Aa
<a+A> <A-A>;"<BASE><VRNT1>";"<MIN><CAP>";"<U0061><U0041>" % aA
<a+a> <A-A>;"<BASE><VRNT1>";"<MIN><MIN>";"<U0061><U0061>" % aa


% The letters u with diaresis 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>";<U00DC> % Ü WITH DIAERESIS
<U00FC> <S0079>;"<BASE><VRNT1>";"<MIN><MIN>";<U00FC> % u WITH DIAERESIS
<U0170> <S0079>;"<BASE><VRNT2>";"<CAP><MIN>";<U0170> % U WITH DOUBLE ACUTE
<U0171> <S0079>;"<BASE><VRNT2>";"<MIN><MIN>";<U0171> % u WITH DOUBLE ACUTE

% And replicate the weighting for the collating-element's formed with combining accents
<U+combining-diaeresis> <S0079>;"<BASE><VRNT1>";"<CAP><MIN>";"<U0055><U0308>"
<u+combining-diaeresis> <S0079>;"<BASE><VRNT1>";"<MIN><MIN>";"<U0075><U0308>"
<U+combining-doubleacute> <S0079>;"<BASE><VRNT2>";"<CAP><MIN>";"<U0055><U030B>"
<u+combining-doubleacute> <S0079>;"<BASE><VRNT2>";"<MIN><MIN>";"<U0075><U030B>"

% 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><CAP>";<U00D0> % ETH
<U00F0> <S0064>;"<BASE><VRNT1>";"<MIN><MIN>";<U00F0> % 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><COMPAT><CAP>";<U00DE> % THORN
<U00FE> "<S0074><S0068>";"<BASE><VRNT1><BASE>";"<MIN><COMPAT><MIN>";<U00FE> % thorn

% The letter oe is treated as a sequence of o + e, with a variant weight
% at the secondary level
reorder-after <U006F> % this *line* only given for stability
<U01D2> "<S006F><S0065>";"<BASE><VRNT1><BASE>";"<CAP><COMPAT><CAP>";<U01D2> % OE
<U01D3> "<S006F><S0065>";"<BASE><VRNT1><BASE>";"<MIN><COMPAT><MIN>";<U01D3> % 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>";<U002D> % HYPHEN-MINUS
<U2010> <S0020>;"<BASE><VRNT1>";"<MIN><MIN>";<U2010> % HYPHEN
<U002F> <S0020>;"<BASE><VRNT2>";"<MIN><MIN>";<U002F> % 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>";<U0138> % kra

% The letter sharp s is treated as a sequence of s + s
% with a variant weight at the secondary level to make it come
% before s+s - shorter precedes longer.
reorder-after <U0071> % q - this *line* only given for stability
<U00DF> "<S0073><S0073>;"<BASE><VRNT1><BASE>";"<RES-1><RES-1>";<U00DF> % 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
ĆB	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

Annex C – Preparation (informative)

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 e.g. is needed for Lao and Thai (see Annex C.2), or proper ordering of numerals (see Annex C.3). A duplicating preparation can map a given string to several strings (to be sorted).

Examples of non-duplicating preparations are:

- 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:

- 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 – a case involving preparation required to get the proper ordering

C.2.1. Thai Ordering Principle

The widely accepted standard for Thai lexicographical ordering is defined in the Royal Institute Dictionary 2525 B.E. Edition (1982 A.D.), the official standard Thai dictionary. The ordering principles are:

- 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 ฤๅ ล ฦ ฦๅ ว ศ ษ ส ห ฬ อ ฮ

(ฤ ฤๅ ฦ ฦๅ are vowels and ligatures, but put in the order according to the sounds they represent.)

- Vowels are also ordered by written forms, not by sounds. Vowels order is:

ะ ั า ำ ิ ี ึ ื ุ ู เ แ โ ใ ไ

(อ ว ย are always ordered as consonants, although they sometimes act as vowels.)

- String comparison is performed from left to right, but considering initial consonants before vowels in the same syllable.
- Tones and diacritics are ignored at level 1, at level 2 the order is:

็ ่ ้ ๊ ๋

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	จุ๋มพล	ฤทธิ์
กัก	ขนาบ	ชาย	ฤษี
ก้าว	ข้าง	เฒ่า	ฤาษี
กำ	ข้างๆ	เณร	ลลิตา
กิน	ข้างกระดาน	ตลาด	ฦาชา
กี่	ข้างขึ้น	ทูลเกล้า	วก
กี๋น	ข้างควาย	ทูลเกล้าฯ	ศาล
กุน	ข้างๆ คูๆ	ทูลเกล้าทูลกระหม่อม	หริภุญชัย
กูด	ข้างเงิน	น้ำ	หฤทัย
เก้ง	ข้างออก	น้ำ๋	หลง
เกล้า	เขน	นี้	แห่ง
เกลียว	เข็น	บุญหลง	แห่ง
เก้า	เข่น	บุญ-หลง	แหนม
เกาะ	เข็ด	ป่า	แหนหวง
เกี่ยว	แข็ง	ป่า	แหบ
เกี๊ยะ	แข่ง	ป้า	แหม
เกือก	แข้ง	ป๊า	อาน
แกง	แข้งขวา	ป๋า	ฮา
แกะ	แข็งขัน	ปาน	

C.2.2 Algorithmic Aspect

These are the only two mandatory requirements for Thai string collation algorithms.

- Leading vowels (เ- แ- โ- ใ- ไ-, corresponding to characters U+0E40-U+0E44), which are written before consonants, must be considered after the initial consonant. Therefore, the rearrangement is needed before comparison.
- Diacritics and tone marks (◌ั ◌ิ ◌ี ◌ึ ◌ื ◌ุ ◌ู ◌ฺ ◌็ ◌่ ◌้ ◌๊ ◌๋ ◌์) must be ignored in the first pass, and be considered at later pass if the first pass yields equality.

No syllable structure or word boundary analysis is required, as Thai lexicons are ordered alphabetically, not phonetically.

C.2.2.1. Leading Vowel Rearrangement

To fullfill this requirement, either a collation preparation or collating-element grouping is required. The collation preparation scans the string once and swaps every leading vowel with its succeeding letter. The prepared string is then passed to the normal weight calculation process. Another way to manage this is by means of collating-element formation. Every possible pair of leading vowel and consonant is defined as a collating-element, whose weight equals to that of the rearranged substring.

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. For example:

1. 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.

2. 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.2.2.2. The Multiple Levels of Character Weights

The second requirement of the algorithm, relating to the treatment of diacritics and tone marks, implies multiple levels of weights. Tone marks and diacritics must be ignored in the first level, and weigh more than consonants and vowels in the second level.

There are 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 digits in the first level, and are different in the second level, to distinguish languages.

When punctuation marks (ฯ ๆ ฿ ◉ ฯ ◌๎) 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 must be ignored in all the first three levels.

For example, "ข้างๆ, ข้างกบ, ข้างๆ คูๆ, ข้างจันˮ is a valid order in the Royal Institute Dictionary. In the first level, the considered weights are ขาง, ขางกบ, ขางคู, ขางจัน respectively.

The third level is not needed for Thai string ordering, but is reserved for tailoring.

C.3. Handling of numeral substrings in collation

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.3.1. 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 the this International Standard 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:

Release 1
Release 2
Release 9
Release 12
Release 20

The Common Template Table defined in this International Standard 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 following will result:

Release 001
Release 020
Release 012
Release 002
Release 009

Using the Common Template Table defined in this International Standard 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 must 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 International Standard 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 standard 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 International Standard.

C.3.2. 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.3.3. 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. E.g. “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.3.4. 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 U+002D (but this character may often represent a true hyphen, rather than a negation), or MINUS SIGN U+2212. However, there are other conventions also, like using a SOLIDUS U+002F or a PERCENT SIGN U+0025 to indicate negativeness; or the negation indicator can come *after* the digits rather than before; or negativeness 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 negativeness is indicated by an immediately prepended MINUS SIGN is dealt with. Positiveness is indicated by either the absence of a MINUS SIGN, or the presence of a PLUS SIGN U+002B.

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):

1. 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.

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

3. 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.

4. 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).

5. Do 9's complement on each digit in each copy of a negated numeral. 9's complement of a digit that individually represents the value *x*, is 9–*x*. That is, 9's 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 standard, 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: –9785 °C –14
Temperature: +015 °C 05
Temperature: +015 °C +5
Temperature: –99 °C –0
Temperature: –980 °C –09
Temperature: +03105 °C 105
Temperature: +015 °C +05
Temperature: +015 °C 5

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

Temperature: –9785 °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: +03105 °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 below).

C.3.5. Handling of positive positional numerals with fractional parts

The method presented above 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. The separator character is generally either FULL STOP U+002E, or COMMA U+002C. 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 above, 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.3.6. Handling of positive positional numerals with fraction parts and exponent parts

For very large, or very small, values, one often uses formats like $2.5*10^7$ (to illustrate just one possible way of writing these for the purposes of the examples here). Here there is already an exponent, which must 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 $2.5*10^{-7}$ we get +49425. Negative values are handled as before, with 9's complement. $–2.5*10^7$ gives –49174, and $–2.5*10^{-7}$ gives –50574. This method should be familiar to anyone with knowledge about (radix 10) floating point arithmetic.

Thus:

$2.5*10^{-7}$
$–2.5*10^7$
$2.5*10^7$
$–2.5*10^{-7}$

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

+49425 $2.5*10^{-7}$
–49174 $^{-}2.5*10^7$
+50825 $2.5*10^7$
–50574 $^{-}2.5*10^{-7}$

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.3.7. 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 above.

1. 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.

2. 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, must 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.

3. 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.

4. 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.

5. 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.

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

7. Use full year number representation, as many digits as needed. Take abbreviations into account so that the full year number is used. E.g. '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.

8. 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.)

9. 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.

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

11. 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.)

12. 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 standard. 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.3.8. 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 must pay attention not to map different strings to identical strings (or identical string fields), so that determinacy is maintained (see C.3.9).

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, e.g., 1,2-diclorobenzol.

C.3.9. Maintaining determinacy

As noted above 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 International Standard 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 International Standard, however.

Maintenance of determinacy when some of the original strings to be collated are identical, is out of the scope of this International Standard. 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.

Annex D – Tutorial on solutions brought by this standard to problems of lexical ordering (informative)

Why does not existing standard character codes, with character-by-character comparisons, give appropriate resulting ordering of strings? What must be done in order to get appropriate resulting orderings? In this discussion will illustrate this with examples the Latin script.

D.1. Problems

1. Sorting, in any language using the Latin script, including English, using, e.g., ISO/IEC 646 coding, does not follow traditional dictionary sequence, which is the minimum the average user needs.

   For example: Ordering the list "august", "August", "container", "coop", “co-op", "Vice-president", "Vice versa" gives the following order, if ISO/IEC 646 coding is used and a simple sort following binary order is performed:

   ```
   August
   Vice versa
   Vice-president
   august
   co-op
   container
   coop
   ```

   This ordering is obviously incorrect.

2. Transforming uppercase to lowercase and removing special characters yields a sorted list acceptable to users, but also yields unpredictable results.

   For example: Sorting the list "August", "august", "coop", "co-op" gives the following order:

   ```
   August
   august
   coop
   co-op
   ```

   Sorting the same list with a different initial order, say, "august", "co-op", "August", "coop" may give a different order with this method:

   ```
   august
   August
   co-op
   coop
   ```

3. If accented characters are introduced using for example any ISO/IEC 8-bit character set, the same problems as encountered above are amplified but they share the same causes.

4. If character code point tables were reorganized to make all related characters contiguous, one might think that a simplified single-character sort would result, but this does not work either. Take upper- and lowercase unaccented letters as an example. If code position 01 is assigned to "a", code position 02 assigned to "A", code position 03 to "b", code position 04 to "B" and so on, a list sorted directly according to these rearranged values will yield the following:

Sorted List	**Internal Values**
aaaa	01010101
abbb	01030303
Aaaa	02010101
Abbb	02030303

   This is also predictable, but remains obviously incorrect for any country with regard to cultural expectations.

D.2. Solution

The only solution to the above problems is to consider the initial data in multiple levels in a way that will respect traditional lexical order, and at the same time ensure absolute predictability. For the Latin script, this can be achieved by comparing in four (or more) levels:

1. The first level renders the texts to be sorted case-insensitive and insensitive to diacritical marks, and to all special characters (which have no pre-established traditional order).

   An example for English:

   "résumé" (’curriculum vitae’) becomes "resume" (‘begin again’), without any accent.

   An example for French:

   "Vice-légation" becomes "vicelegation", with no accent, no uppercase and no hyphen.

   An example for German:

   "groß" becomes "gross", with the sharp-s being converted to double-s for the ordering.

   In some languages, including Spanish and the Nordic languages, some extra letters are added to the 26 letters of the English, French, and German alphabets. The additional letters are not ordered according to the expectations in other languages. This demonstrates the need for adaptability.

2. The second level breaks ties on quasi-homographs, that is, strings that differ only because they have different diacritical marks.

   In English, "resumé" and "résumé" are quasi-homographs. Traditional English lexical order requires that "resume" always comes before "résumé" (which sorting using only the first level would not guarantee). In this case, the tradition does not explicitly specify whether "resumé" should come before "résumé", though this would seem logical: most English and German dictionaries only state that unaccented words precede the accented words. However, German dictionaries generally employ the German standard DIN 5007, which states rules that are more precise.

   Here another characteristic is introduced. In French, because of the large number of multiple quasi-homograph groups formed of more than 2 instances, the most important dictionaries follow the following rule: Accents are generally not taken into account for sorting, but in case of homographic ties, the *last* difference in the word determines the correct order between two given words. A priority order is assigned to each type of accent. According to this, "coté" should be sorted after "côte" but before "côté". This is easy to implement with “backwards” tailoring: A number is assigned to each character of the data to be sorted, representing either a letter with an accent or a letter with no accent at all, but these numbers are prepended instead of being appended to the result being constructed. In other words, the resulting string is made starting from the last character of the original data and processing in a backwards direction.

   For example: To obtain an order respecting this rule: "cote, "côte", "coté", "côté", numbers could be assigned indicating respectively "****", "**C*", "A***", "A*C*", where "*" means no accent, "A" means acute accent, "C" circumflex accent. This scheme is sufficient to break the tie correctly at this second level.

3. The third level breaks ties for quasi-homographs that differ only because uppercase and lowercase characters are used.

This time, the tradition is well established in German dictionaries, where lowercase always precedes uppercase in homographs, while the tradition is not well established in French dictionaries, which generally use only accented capital letters for common word entries. In known French dictionaries where upper- and lowercase letters are mixed, the capitals generally come first, though this is not an established and stated rule, because there are numerous exceptions. English has no monolithic practice for this, a bit like French. Therefore, for a Common Template it is advisable to use the well-established German tradition, if one wants to group the largest possible number of languages together without affecting others. Note that in Denmark, uppercase is specified to precede lowercase, a different but well-established rule. This is a second fact that demonstrates the need for adaptability in the model used in this International Standard.

For example: to have the following order: "august", "August", numbers could be assigned indicating respectively "LLLLLL", "ULLLLL", where "L" means lowercase and "U" uppercase.

4. The fourth level breaks the final tie that, in general, does not correspond to any strong tradition, namely, the tie between quasi-homographs differing only because they contain special characters.

   Breaking this tie is essential to ensure the predictability of ordering as well as enabling the ordering of strings composed only of special characters. Since the traces of special characters were removed from the original data to form the three first levels, simply putting them sequentially in the fourth order of decomposition would mean that their position would be lost. These positions are needed to resolve remaining ties, so the original positions of these special characters must be retained somehow. E.g., two quasi-homographs could each contain a common special character in different positions and thus be strictly different (example: "ab*cd" is different from "a*bcd" despite they share one and only one common special character).

   For example: To obtain the following order: "coop", "co-op", "coop-", numbers could be assigned respectively according to the following pattern: "", "3-", and "5-"; where "3-" means a hyphen in (relative) position 3 of the original string. "5-" means a hyphen in (relative) position 5, and so on. Note that "coop.", "co-op.", "coop-." (adding a period at the end of each string) get the numbers assigned as "5.", "3-3.", and "5-1.", and are thus ordered: "co-op.", "coop.", "coop-.".

These four levels can be composed to a four-level key, concatenating the subkeys from the most significant to the least significant, putting the lowest value possible as a delimiter between each subkey. The ordering result can then be obtained through the numeric order of the keys.

If the assignment of weight numbers is done properly, one can eliminate some of the delimiter weights. To eliminate the level delimiter between the first and second level subkeys, choose the numbers for the second level weights be less than numbers for the first level weights. To eliminate the level delimiter between the second and third level subkeys, choose the numbers for the third level weights be less than numbers for the second level weights.

Subkey reduction techniques have been designed to considerably shorten space requirements. As no implementation is required to use specific numbers for weights and reduction is not required, this issue is outside the scope of this International Standard. Nevertheless, it is interesting to note that implementation can be optimised. This has been improved over time and is easy to accomplish, some methods being more efficient than others. Annex E below describes two example order-preserving subkey reduction techniques.

This method for string collation was described with tables in *Règles du classement alphabétique en langue française et procédure informatisée pour le tri*, Alain LaBonté, Ministère des Communications du Québec, 1988-08-19, ISBN 2-550-19046-7. A public-domain subkey reduction technique is described (with several examples) in *Technique de réduction - Tris informatiques à quatre clés,* Alain LaBonté,

Ministère des Communications du Québec, 1989-06, ISBN 2-550-19965-0. See also the paper by Rolf Gavare, *Alphabetic ordering in a lexicological perspective*, Studies in Computer-Aided Lexicology, 1988, pp. 63–102, which also describes a multi-level string collation technique, with subkey compression.

D.3. Tailoring

For a number of languages, the Common Template Table presented in this standard will need to be adapted. Adaptation may be needed both in the table values for the four levels of subkeys, which can require redefining weightings for characters or introducing multi-character collating elements into the table; as well as changes in the potential context analysis processing necessary to achieve culturally correct results for users of these languages.

To illustrate this, without discussing context analysis which is not necessary in what follows, examples of dictionary sequences are given here for two languages whose expected ordering rules are not covered the Common Template Table:

For traditional Spanish, where "ch" is greater than "cu" and "ña" is greater than "no":

cuneo < cúneo < chapeo < nodo < ñaco

Comparative French/English/German sort of the same strings:

chapeo < cuneo < cúneo < ñaco< nodo

For Danish, where "a" is less than "c", "cz" is less than "cæ" and "cø", and "aa" is equivalent to "å", which is greater than "z", even in cases where it is pronounced differently:

Alzheimer < czar < cæsium < cølibat < Aachen < Aalborg < Århus

Comparative French/English/German sort of the same strings:

Aachen < Aalborg < Alzheimer < Århus < cæsium < cølibat < czar

Similarly, Japanese will need tailoring in order to handle the length mark properly; Thai and Lao will need preparation or tailoring to handle vowel-consonant rearrangement properly (see C.2). For may other orthographies, some degree of tailoring will be necessary.

Annex E – Order-preserving subkey reduction (informative)

The template table of collation weights has four levels. When applied to a string, each level nominally produces a subkey that is about as long, in number of weights, as the number of characters in the string itself. There are, however, a number of ways to reduce the size of the subkeys without changing the ordering as determined by the nominal key. This international standard neither specifies normatively, and even less requires the use of, any subkey reduction technique. However, for conformity, any key size reduction method must preserve the order between strings as determined by the nominal key produced by the given tailoring of the template table.

In order to illustrate what can be done in terms of subkey reduction, we here present two example reduction methods. Good implementations of these methods produce the reduced key directly, without producing the nominal key first.

Applied correctly to each nominal key, these reduction methods keep the order between the strings, since each is a strictly monotonically increasing mapping, i.e.,

if *nominal_key1* < *nominal_key2*, then
reduce1(*nominal_key1*) < **reduce1**(*nominal_key2*), and
reduce2(*nominal_key1*) < **reduce2**(*nominal_key2*).

Each subkey reduction method that results in a strictly monotonically increasing mapping can be applied at any level. Moreover, different reduction methods can be applied at different levels, as long as it is done consistently for all keys. E.g., example method 1 below can be used for levels 2 and 4, while using example method 2 below for level 3, while no reduction method need be applied to level 1.

These example methods are not normative, and can probably be improved upon to reduce the subkey sizes further (e.g., one may first remove trailing weights that are minimal at levels 2 and 3).

E.1. Example subkey reduction method 1: interleaved counts and weights

This method can be applied for a single selected weight at each level this method is used, preferably a weight that is very commonly occurring at that level.

This method uses a count value, which may be stored in the subkey using different number of digits from what the weights uses at that level. The transformed value (see below) of the count need be not related to any of the weight values. In this illustration, we will use integer values between 00 and FF (in hexadecimal), the latter is the $maxcnt_q$. A practical implementation may of course use a wider range of values.

The reduction method works as follows, described in principle, not in implementation terms: Each maximal subrun of the for that level selected weight to be reduced, including subruns of length 0, is replaced by a transformed count value, as follows:

- Consider each subkey to be terminated by a less than minimal (at that level), or zero, weight.
- swa_q is the selected reduction weight at level q.
- *wb* is the weight (including subkey terminator weight) that follows the (empty or not) maximal subrun, of the weight swa_q, currently processed.
- *cnt* is the length (≥ 0) of the subrun currently processed.
- $maxcnt_q$ is the value (≥ 0) that is the largest value that can be stored as a count in the subkey, given the number of digits used for the storage of a (transformed) count in a subkey at that level.
- $guardcnt_q$ is a value ≥ 0 and ≤ $maxcnt_q$ (e.g. $maxcnt_q$ **div** 2 can be used as $guardcnt_q$).
- $lowcntsize_q = guardcnt_q + 1$.

- $highcntsize_q = maxcnt_q - guardcnt_q + 1$.
- If $swa_q > wb$: Replace the subrun with the count/weight-pair ($guardcnt_q$, swa_q) repeated (*cnt* **div** $lowcntsize_q$) times, followed by the count (*cnt* **mod** $lowcntsize_q$).
- If $swa_q < wb$: Replace the subrun with the count/weight-pair ($guardcnt_q$, swa_q) repeated (*cnt* **div** $highcntsize_q$) times, followed by the count ($maxcnt_q$ − (*cnt* **mod** $highcntsize_q$)).

Note that if the limits are set reasonably high, the count/weight-pairs will be repeated zero times in practical cases. In the other extreme, if $maxcnt_q$ is zero, the subkey is not changed.

The reason this method works only for one weight per level is that subruns otherwise would cause ambiguous key reductions.

This reduction method can be applied to level 2, where <BASE> can be expected to be a very common weight. It can also be used for level 3, where <MIN> (minuscule, i.e. lowercase, or caseless) can be expected to be very common. This reduction method can also be applied to level 4, if one uses a tailoring that gives one very common weight on level 4. Note that if the selected weight is not very common in a string, the resulting key by this method may be longer than the nominal key, since empty subruns of the selected weight must be replaced by another subrun that is non-empty.

Below is an example reduction using this method on level 2 (selecting the weight <BASE>), on level 3 (selecting the weight <MIN>), and on level 4 (selecting a maximal weight called <PLAIN>; for a tailoring of the template table). For readability, we will in this example write `<a>`, `<b>`, etc. for the weights of the letters, `<->` for <BLANK>, `<l>` for <MIN>, `<u>` for <CAP>, `<*>` for <PLAIN>, `<A>` for <ACUTE>, `<H>` for the level 4 weight of a hyphen, and `<P>` for the level 4 weight of an apostrophe. The example character string “Vice-Président’s”. The nominal key, according to the template table, can be expressed as a number in *R*, expressed as a hexadecimal fractional number; <v> etc. really stands for digit sequences; spaces are used for alignment for clarity, they are in no way part the actual key; the 0000 at the end of the subkeys is the subkey termination weight):

```
0.<v><i><c><e>   <p><r><e>   <s><i><d><e><n><t>   <s> 0000
  <-><-><-><->   <-><-><-><A><-><-><-><-><-><->   <-> 0000
  <u><l><l><l>   <u><l><l>   <l><l><l><l><l><l>   <l> 0000
  <*><*><*><*><H><*><*><*>   <*><*><*><*><*><*><P><*> 0000
```

Do the reduction as described above (note that here: `<->` (and `<l>`) is smaller than any other level 2 (level 3) weight, but greater than 0 (the subkey termination weight), and `<*>` is greater than any other level 4 weight):

```
0.<v><i><c><e>   <p><r><e>   <s><i><d><e><n><t>   <s>   0000
  F8                      <A> 07                         0000
  FF <u>       FC <u> 0B                                 0000
  04          <H> 09                           <P> 01   0000
```

Note that “count” values are at the beginning and end of the reduced subkeys, as well as between each non-selected weight. This key is significantly shorter than the nominal key, in this example as well as for most (not all) other strings that one can expect to normally occur.

E.2. Example subkey reduction method 2: each count integrated in a weight

This method can be applied for a set of weights at each level, preferably the ones that are commonly used at that level.

This method also uses a count value, but the transformed weight value must have the same number of digits as all other weights at that level uses. The transformed values (see below) for the count must have values that are in the neighborhood of the weight in the subrun that is replaced. This neighborhood of a weight must not overlap with any other weights or neighborhoods of weights.

The reduction method works as follows, described in principle, not in implementation terms: Each non-empty maximal subrun of a weight selected for reduction is replaced as follows:

- Consider each subkey to be terminated by a less than minimal (at that level), or zero, weight.
- *wa* is the weight in the subrun.
- *wb* is the weight (including subkey terminator) that follows the non-empty maximal subrun of *wa* currently processed.
- *cnt* is the length (≥ 1) of the subrun currently processed.
- $minnbh_{wa}$ is the minimum value that constitutes the neighborhood of *wa*.
- $maxnbh_{wa}$ is the maximum value that constitutes the neighborhood of *wa*.
- $lownbhsize_{wa} = wa - minnbh_{wa} + 1$.
- $highnbhsize_{wa} = maxnbh_{wa} - wa + 1$.
- If *wa* > *wb*: Replace the subrun with the weight *wa* repeated (*cnt* **div** $lownbhsize_{wa}$) times, and, if (*cnt* **mod** $lownbhsize_{wa}$) is non-zero, followed by the weight ($minnbh_{wa}$ + (*cnt* **mod** $lownbhsize_{wa}$) − 1).
- If *wa* < *wb*: Replace the subrun with the weight *wa* repeated (*cnt* **div** $highnbhsize_{wa}$) times, and, if (*cnt* **mod** $highnbhsize_{wa}$) is non-zero, followed by the weight ($maxnbh_{wa}$ − (*cnt* **mod** $highnbhsize_{wa}$) + 1).

Note that if the neighborhood of *wa* consists of *wa* only, the subrun is not changed.

This reduction method can be applied to level 2, for <BASE>. It can also be used for level 3, e.g. for <MIN>, <CAP>, <HIRA>, and <KATA>. It can also be applied to level 4, for <PLAIN> (for a tailoring that uses <PLAIN> as a maximal weight at level 4). Note that even if the weight reduced is not common, the resulting subkey is never longer than the nominal subkey. A nontrivial neighborhood is needed around each of the selected weights for a shortening of the subkey to actually take place.

We do an example reduction using this method on level 2 (for <BASE>), on level 3 (for <MIN> and <CAP>), and on level 4 (for <PLAIN>). For this we make the following assumptions:

- `<->` (level 2): weight: 0026; minimum – maximum of neighborhood: 0022 – 002A.
- `<l>` (level 3): weight: 0005; minimum – maximum of neighborhood: 0002 – 0007.
- `<u>` (level 3): weight: 0017; minimum – maximum of neighborhood: 0015 – 001A.
- `<*>` (level 4): weight: 0F80; minimum – maximum of neighborhood: 0F00 – 0FFE.

For the same example string as in the description of example reduction method 1, we still have the nominal key:

```
0.<v><i><c><e>   <p><r><e>   <s><i><d><e><n><t>   <s>   0000
  <-><-><-><->   <-><-><-><A><-><-><-><-><-><->   <->   0000
  <u><l><l><l>   <u><l><l>   <l><l><l><l><l><l>   <l>   0000
  <*><*><*><*><H><*><*><*>   <*><*><*><*><*><*><P><*>   0000
```

Do the reduction as described above:

```
0.<v><i><c><e>   <p><r><e>   <s><i><d><e><n><t>   <s>   0000
  0024                    <A>0028                       0000
  0015  0005     0015  0005 0005 0002                   0000
  0F03         <H>0F08                          <P>0F00  0000
```

If the lower neighborhood of `<l>` had been a bit bigger, we could have used only one weight (or two), instead of three, for the sequence of nine `<l>` weights.

This key is also significantly shorter than the nominal key, and this method never lengthens the key, since every subrun replaced is replaced by one that is shorter or at most as long as the original subrun. This method can also be applied to several weights at each level, which example method 1 cannot. On the other hand, example method 2 is a bit more complex than example method 1, since it needs to keep track of non-overlapping neighborhoods around some of the weights.

E.3. Remapping of weights to a common weight for better subkey reduction

In many cases, at level 2 and up, differing weights don't convey a real difference for the ordering, since subkeys that are more significant may contain a necessary ordering difference already. This is especially common for the fourth level, where the Common Template Table has different weights for every collating element. Since there are usually other more significant differences, it is rarely this difference at the fourth level matters. Implementers are in such cases allowed to map several nominally different weights to the same actual weight, in such a way that the resulting ordering is unchanged, in order to enable better subkey reduction.

Annex F – Bibliography (informative)

The following standards and documents are considered relevant to this standard, in addition to the normative references.

- CAN/CSA Z243.4.1-1998 – Canadian Alphanumeric Ordering Standard, A National Standard of Canada, Canadian Standards Association.
- CAN/CSA Z243.230-1998 – Minimum Canadian Software Localisation Conventions, A National Standard of Canada.
- DS 377:1980 – Alfabetiseringsregler, Dansk Standard.
- Gavare, Rolf, Alphabetic ordering in a lexicological perspective, Studies in Computer-Aided Lexicology, 1988, pp. 63–102.
- ISO/IEC 646, Information technology – ISO 7-bit coded character set for information interchange.
- ISO/IEC 2022, Information technology – Code extension techniques.
- ISO/IEC 6937, Information technology – Coded character sets for text communication.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 2: Latin alphabet No. 2.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 3: Latin alphabet No. 3.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 4: Latin alphabet No. 4.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 5: Latin/Cyrillic alphabet
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 6: Latin/Arabic alphabet.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 7: Latin/Greek alphabet.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 8: Latin/Hebrew alphabet.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 9: Latin alphabet No. 5.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 10: Latin alphabet No. 6.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 13: Part 13: Latin alphabet No. 7.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 14: Latin alphabet No. 8.
- ISO/IEC 8859-1, Information technology – 8-bit single-byte coded graphic character sets – Part 15: Latin alphabet No. 9.

- ISO/IEC 9945-2, Information Technology – Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities.
- ISO/IEC DTR 14652, Information Technology – Specification Method for Cultural Conventions.
- Règles du classement alphabétique en langue française et procédure informatisée pour le tri, Conseil du trésor du Québec, URL: http://www.tresor.gouv.qc.ca/doc/classm.htm.
- Retskrivningsordbogen – 2nd edition 1996, Dansk Sprognævn & Aschehoug Dansk Forlag A/S.
- Technique de réduction - Tris informatiques à quatre clés, Conseil du trésor du Québec, URL: http://www.tresor.gouv.qc.ca/doc/techtri.htm.
- Teknisk norm nr. 34, Swedish Alphanumeric Sorting, Statskontoret, 1992. (Includes Gavare's paper as an annex.)
- The Unicode Standard, Version 2.0, The Unicode Consortium, Addison Wesley Developers Press, ISBN 0-201-48345-9.
- Unicode Technical Report no. 8, The Unicode Standard, Version 2.1, The Unicode Consortium, URL: http://www.unicode.org/unicode/reports/tr8/.
- Unicode Technical Report no. 10, Unicode Collation Algorithm, The Unicode Consortium, URL: http://www.unicode.org/unicode/reports/tr10/.
- Unicode Technical Report no. 15, Unicode Normalization Forms, The Unicode Consortium, URL: http://www.unicode.org/unicode/reports/tr15/.