ISO/IEC JTC1/SC 22/WG 20 N ~~619~~665

Date~~: 16 November 1998~~1999-04-21

ISO
ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
Ì ÅÆÄÓ Í ÀÐ Î Ä Í ÀÀß ÎÐÃÀ Í ÈÇÀÖÈß Ï Î ÑÒÀ Í ÄÀÐÒÈÇÀÖÈ È

CEI (IEC)
COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE
INTERNATIONAL ELECTROTECHNICAL COMMISSION
Ì ÅÆÄÓ • € • Ž„ Í ÀÀ ÇËÅÊÒÐ Î ÒÅÕ Í È × ÅÑÊÀß Ê Î Ì ÈÑÑÈß

Title **Proposed editorial changes to ISO/IEC FCD 14651.2** - International String Ordering - Method for comparing Character Strings and Description of the Common Template Tailorable Ordering

[ISO/CEI CD 14651 - Classement international de chaînes de caractères - Méthode de comparaison de chaînes de caractères et description du modèle commun d'ordre de classement]

Status: ~~Final Committee Document~~Expert contribution

Reference: SC22/WG20 N ~~568R (Disposition of comments on first FCD ballot)~~619

Date: ~~1998-11-16~~1999-04-21

Project: 22.30.02.02

Editor: ~~Alain LaBonté~~

~~Gouvernement du Québec~~
~~Secrétariat du Conseil du trésor~~
~~875, Grande-Allée Est, Secteur 3C~~
~~Québec, QC G1R 5R8~~
~~Canada~~Michael Everson

Email: ~~alb@sct.gouv.qc.ca~~everson@indigo.ie

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 for ordering text data worldwide, and provides a Common Template Table whose tailoring eases adaptation of a specific script to the requirements of a given language while retaining universal properties for other scripts. The purpose of such a mechanism is to correct past errors ~~of the past~~ regarding collation ~~done~~, since previously collation operations were based only on binary coded character values. Past approaches have often not respected cultural preferences for collation. ~~English is one exception, although a poor one, when only upper case alphabetic data was used instead of other characters including punctuation and spacing.~~

This is one of the major flaws ~~that affect~~ affecting portability between countries and between applications. (Traditionally, different programs used different ordering specifications.) Therefore, it has been considered ~~feasible~~ necessary to design a Common Template Table for ordering and a comparison method that can be used as a reference for the results to be achieved by implementations. This Standard is the ~~achievement~~ result of efforts to meet this challenge.

The Common Template Table requires some tailoring in different local environments. However conformance to this International Standard requires that ~~the~~ all deviations ~~to~~ from the Template, called "deltas", be declared to document result discrepancies.

This Standard describes a method to order text data independently of context. It has provisions to allow a ~~lot~~ great deal of of flexibility in implementations, while remaining an excellent international reference for ordering to which all technical and cultural environments can commonly refer ~~to~~.

1 Scope

This International Standard defines:

- A simple method of reference for comparing two characters strings in order to determine their respective order in a sorted list. The method ~~is applicable~~ can be applied on strings ~~that~~ exploiting the full repertoire of ISO/IEC 10646 (independently of coding). These comparisons are also applicable ~~for~~ to ~~subrepertoires~~subsets of that repertoire, such as those of the different parts of ISO/IEC 8859 parts or any other character set, standardized or private, to produce ordering results valid (after tailoring) ~~in~~ for a given set of languages for each script. This method uses transformation tables derived either from ~~either~~ the Common Template Table defined in this International Standard or from one of its tailorings.

- A reference format using a variant of the Backus-Naur Form (BNF) to describe the Common Template Table used normatively in this International Standard.

- A specific Common Template Table used by the comparison method. This table describes a basic order for all characters ~~specified~~ encoded in the first edition of ISO/IEC 10646-1 up to amendment ~~7 in this edition of this International Standard~~31. ~~This~~ The table is a starting point ~~in~~ for enabling the specification of culturally acceptable orders adapted to different cultures, without requiring an implementor to have a knowledge of all the different scripts already ~~taken care of by this International Standard~~encoded in the UCS.

  ~~Note~~*NOTE: It is to be considered normal practice that this Common Template Table be modified with a minimum of ~~efforts~~ effort to suit the needs of a local environment. The main benefit, worldwide, is that for other scripts, no modification ~~is~~ should be required and that the order will remain as consistent as possible and predictable from an international point of view.*

- A reference name representing this particular version of the Common Template Table for use by various applications as a starting base for 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.

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 the name of the Common Template Table.

- A specific user interface for choosing options.

- A specific internal format for intermediate keys used in comparisons nor for the table used. The use of numeric keys is not mandated either.

- A context-dependent ordering which would require complex transformation of data to order.

Note: Although no user interface is ~~prescribed~~ 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 highly recommended that these fundamental choices be presented by the application interfacing with the users of the results produced, for instance, in a Preferences dialogue box.

2 Conformance

An application is conformant to this International Standard if it meets the requirements prescribed in ~~section~~ clause 6.

~~Any~~ A declaration of conformity to this International Standard shall be accompanied by a declaration of the tailoring *delta* described in clause 6.4 in case ~~tailoring is not provided by the concerned~~ the application concerned does not provide a tailoring facility to the end user. In the case that such tailoring is provided, the declaration shall indicate which of the elements of clause 6.3 it is possible to tailor in the ~~concerned~~ application and which ~~ones~~ elements have no tailoring provisions. More specifically, it is the responsibility of ~~implementers~~ implementors to show how their delta declaration is related to the table syntax described in clause 6.3, and how the comparison method they use, if ~~, If~~ different from the one mentioned in clause 6.1, can be considered as giving the same results as those prescribed by the method specified in clause 6.1.

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.3:1996 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 3.*

- 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.10:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 10: Ethiopic.*

- ISO/IEC 10646-1:1993/Amd.11:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 11: Unified Canadian Aboriginal Syllabics.*

- ISO/IEC 10646-1:1993/Amd.12:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 12: Cherokee.*

- ISO/IEC 10646-1:1993/Amd.14:1999 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 14: Yi.*

- ISO/IEC 10646-1:1993/Amd.16:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 16: Braille patterns.*

- ISO/IEC 10646-1:1993/Amd.18:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 18: XXX EURO.*

- ISO/IEC 10646-1:1993/Amd.19:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 19: Runic.*

- ISO/IEC 10646-1:1993/Amd.20:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 20: Ogham.*

- ISO/IEC 10646-1:1993/Amd.22:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 22: Keyboard symbols.*

- ISO/IEC 10646-1:1993/Amd.23:1998 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 23: Extended Bopomofo and other characters.*

- ISO/IEC 10646-1:1993/Amd.24:1999 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 24: Thaana.*

- ISO/IEC 10646-1:1993/Amd.25:1999 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 25: Khmer.*

- ISO/IEC 10646-1:1993/Amd.26:1999 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 26: Myanmar.*

- ISO/IEC 10646-1:1993/Amd.27:1999 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 27: Syriac.*

- ISO/IEC 10646-1:1993/Amd.29:1999 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 29: Mongolian.*

- ISO/IEC 10646-1:1993/Amd.30:1999 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 30: Latin and other characters*.

- ISO/IEC 10646-1:1993/Amd.31:1999 *Information technology – Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane Amendment 31: Tibetan.*

4 Definitions

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

4.1 canonical form the coding of a UCS character in 4 octet binary form according to ISO/IEC 10646-1

4.2 character string a sequence of characters considered as a single object

4.3 collation ordering of elements

4.4 collating symbol a symbol used to specify weights assigned to a character

4.5 collating element a single weight or a sequence of weights attributed to a character at a specific level of ordering

4.6 delta list of differences of a specific ordering table relatively to the Common Template Table defined in this International Standard

4.7 glyph a recognizable abstract graphic symbol which is independent of any specific design

4.8 graphic character a character, other than a control function, that has a visual representation normally handwritten, printed, or displayed. To a graphic character normally corresponds a glyph

4.9 level whenever used without qualification in this International Standard, *level* stands for the depth at which a comparison is made on two character strings.

4.10 token a number used as an actual comparison element by the reference comparison method

4.11 ordering a process in which a set of strings are assigned a given order relative to any other set of strings

4.12 ordering key a series of numerical values used to determine an order

4.13 preparation a process in which character strings are modified internally to lead to straightforward comparisons according to this standard

4.14 script a set of graphic characters used for the written form of one or more languages

5 Symbols and abbreviations

Identification of characters of the repertoire of ISO/IEC 10646-1 (Universal multiple-octet-coded Character Set or UCS) ~~repertoire~~ is made in this edition of this International Standard by means of symbols of the form <UXXXX>. The occurrences of XXXX which follow the letter "U" represent the hexadecimal value (using upper case letters when applicable) of a coded character as defined in ISO/IEC 10646 but no specific coded value is intended. What is being referenced is a graphic character, independently of its coding, and any character set ~~whose~~ which is a subrepertoire is ~~taken into account in~~ of ISO/IEC 10646-1 is covered in this way.

This use of symbols is a means to be code-independent (the same value ~~being possibly~~ is used even if the coded character set in use in a given implementation is not ISO/IEC 10646). At the same time, this is a means to keep a straightforward link with the Universal multiple-octet Coded Character Set (UCS), which contains all the coded graphic characters ever defined by ISO/IEC JTC1 standards. Addenda to ISO/IEC 10646 will be published from time to time; these addenda may ~~then also result in~~ require that addenda to this International Standard ~~if necessary~~ also be made.

By convention, if a character outside of the standard repertoire of ISO/IEC 10646 is to be used in tailored ordering tables, it is recommended that the code-independent symbol identifying this character use the form <P*yyyyyyyy*> for documentary purposes indicating its nonstandard nature. The binding to actual coding of these symbols for nonstandard characters is left to implementation and to tailoring. If, for example, actual UCS coding is used, then ~~private~~ Private Use zones of this character set will normally be used for meeting such special requirements, and binding could then be specified so that the sequence *yyyyyyyy* used in the symbol represents ~~private~~ Private Use zone UCS coding.

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

6 Requirements

6.1 Reference method for establishing an order between two character strings

6.1.1 Preparation of character strings prior to comparison

It may be necessary to transform character strings before ~~these character strings are fed into~~ the 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, context-sensitive preparation may be an important part of the ordering process, as for example in telephone-book ordering, a complex case in point.

An application conformant to this ~~international~~ International ~~standard~~ Standard shall, at the minimum, prepare the string so that sequences using either combining sequences or using precomposed characters be presented to the comparison method described in 6.1, if they are meant to be equivalent. An application is also conformant if it uses a comparison method for forming ordering keys which, without making any such preparation, is demonstrated to produce results identical to the ones resulting from the use of such a preparation.

Note 1: In this International Standard, the Common Template Table ~~is built~~ has been constructed so that, with a minimum of effort, precomposed characters will be ordered in the same way as when equivalent sequences of combining characters are used, provided that the preparation prescribed in the previous paragraph ~~is~~ has been made. It ~~is~~ has been demonstrated that by tailoring the Common Template Table to add extra token values at level 2 for all precomposed characters affected by a diacritic~~s~~, it is possible to ~~accomplish~~ obtain identical results for combining sequences without requiring that preparation. However, as it is not typically the case that such double-coding will be used, it is not considered ~~required~~ necessary to add extra tokens, ~~for keeping~~ in order to preserve a reasonable economy of means in the general case.

Note 2: Escape sequences constitute very sensitive data to interpret, and it is highly recommended that preparation should filter out or transform these sequences. Ideally, all control characters should be filtered out before comparison and reintroduced afterward, in case of absolute homography, to distinguish two character strings being compared.

6.1.2 Comparison method of reference resulting in ordering two character strings

The following describes the comparison method used as a reference to determine conformance to this International Standard:

1. In considering a table describing weights at *m* levels for each of *n* characters in the implementation character set, build a numeric key for ~~both~~ each of two arbitrary character strings being compared, according to the algorithm of key formation described in clause 6.2 of this International Standard.

2. Compare the numeric keys produced for the two character strings. The character string whose numeric key is smaller shall be ordered before the other one. If the two numeric keys are identical in value then the two character strings compared shall be considered as equal according to this International Standard.

The table used in this reference method for comparison is the result of the numeric interpretation of the symbols in the Common Template Table, or in a tailored table into a table of *n* by *m* elements (*n* characters by *m* levels described per character). In the Common Template Table, the number of levels *m* described per character is equal to 4.

6.2 Building the ~~Ordering~~ ordering key used in the reference comparison method

6.2.1 Preliminary considerations

6.2.1.1 Assumptions

The ordering table is a transformation table that can be considered as a matrix of *n* lines. *n* is the number of characters in the repertoire used. In each line 4 levels are described in the Common Template Table. This number of levels can be extended in the tailoring phase by the end-user. The user shall ~~take care that, in case of~~ , when tailoring, take are to adjust levels ~~be adjusted~~ so that the last level may processed in a special way according to ~~what is described in what follows~~the description below. Normally the last level is intended to specify "special" characters, i.e., characters normally not part of the ~~orthography of any script~~spelling of words of a language (sich as dingbats, punctuation, etc.), sometimes called "ignorable" characters in the context of computerized ordering.

6.2.1.2 Blocks and processing properties

A tailored table may be separated into blocks. Each block has specific scanning and ordering properties.

One of the tailoring possibilities is to assign a given order to each block and to change the relative order of ~~a whole~~ an entire block relatively to other blocks.

The scanning direction (forward or backward) used to process the string at each level is a property of each block. These properties can be changed.

A specific property of the last level of comparison is that, before comparing weights of each "ignorable" character, a comparison on the numeric position of each such character ~~of~~ in the two strings is effected ~~(in~~ . 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 ignorables share the same positions, then weights are considered, ~~and this~~ until a difference is found).

~~*Note*~~*NOTE: The scanning direction (forward or backward) is not normally related to the natural writing direction of a script or of scripts described by ~~one~~ a particular block. 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 positions in the logical sequence, while scanning backward starts from the highest positions, and this independently of the presentation sequence. The scanning direction for ordering purposes is a property of a block.

In ISO/IEC 10646-1, the Arabic script is artificially separated into two pseudoscripts: 1) the logical, intrinsic Arabic, coded independently of shapes, and 2) the Arabic presentation forms. Both allow to code the complete coding of Arabic completely, but intrinsic Arabic is normally preferred for better processing, while the second presentation-form Arabic is preferred by some presentation-oriented applications (it does not, however, handle most non-Arabic languages using the Arabic script (Sindhi, etc) as full sets of presentation forms for those languages have not been encoded). ISO/IEC 10646-1 does not prescribe that the logical order of the presentation forms be coded in presentation order or logical order. Therefore, tables can be tailored to specify a specific block for these characters and the scanning properties can then be specified according to the coding employed by the application.

6.2.2 Key composition

A series of *m* intermediary subkeys is formed out of a character string composing a comparison field; *m* is the maximum number of levels described in either the Common Template Table or the in the tailored ordering table. The following paragraphs subclauses describe the formation of each of these subkeys whose successive sequence forms a complete ordering key. In the Common Template table, *m* is equal to 4.

6.2.2.1 Formation of subkey level 1 through (m – 1) (level i; m=4 in the Common Template)

For *i* varying from 1 to (*m* – 1) (from 1 to 3 if the Common Template Table is used), form subkey level *i* in the following way:

During forward scanning of each character of the input character string, one or more tokens are obtained. These tokens correspond to the transformation value of that character at level *i*.

The scanning properties for the level *i* being processed needs to must be carefully monitored. When there is a change in scanning direction at level *i* (this implies implying that the character being processed comes from a block that which is different from the preceding character processed and which has different scanning properties) and the new direction is backward, stacking of the token will be done take place at the position where the change of direction has occurred. Therefore, when such a condition occurs, the application shall retain the current position in the output subkey *i* as **position *p*** (*push* position).

According to the scanning direction assigned to the level *i* of the block in which the character being processed belongs, the obtained token is either added in sequence (concatenated) at the end of subkey **i** (which behaves like a list), or pushed at **position *p*** of subkey *i* (which then behaves like a stack). Subkey *i* is initially empty.

Note NOTE: This is the equivalent of backward or forward scanning of the input string for that level. This property of scanning direction is given for each level of each block and is a block property. The Common Template Table has only one block until it is tailored.

6.2.2.2 Formation of subkey level m (m=4 in the Common Template table)

If the `order_start_entry` does not use~~s~~ the ***`position`*** value at level *m* of a block (the ***`position`*** value is explicitly used in the template only for the ~~only~~ block defined) then the formation of subkey level *m* is done in exactly the same way as the ~~above~~-defined formation. Otherwise, the formation of subkey level *m* is as follows, in accordance with ~~frequent~~ common market practice:

> During forward scanning of each character of the input character string, a pair of tokens is concatenated to subkey level *m*. The first token of the pair corresponds to the logical position in the original character string of the character being processed. The second token in the pair corresponds to the weight assigned to that character at level *m* of the table. When the character is not assigned at level *m* in the table, it is ignored for the formation of subkey level *m* and no pair is concatenated. The pair of tokens is concatenated immediately after subkey level *m*. Subkey level *m* is initially empty.

Generally, and in the Common Template Table, levels represent the following decomposition for basic characters:

Level 1: The ~~Base~~ base level of each script. This level corresponds to the set of basic letters of the alphabet for that script, if the script is alphabetic, and to the set of basic characters of the script if the script is ideographic or syllabic.

Level 2: The level corresponding to diacritical marks affecting each basic character of the script. For some ~~scripts~~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 ~~in Spanish~~ 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 ~~a tilde~~ 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 ~~extra~~ a diacritic~~s~~ in the second level".

Level 3: The level corresponding to case or to variant character shape that ~~affects each~~ affecting the basic character~~s~~ of the script.

Level 4: This level represents the level common to all scripts or the level not specifically belonging to any script. The property of this level is that it is ordered positionally according to this International Standard. This means that the numerical value of the position in the original string has precedence over the weight assigned to the special character which occupies this position. ~~This~~ It also means that subkey level *m* is composed of a pair of values for each such character (the character string ~~being~~ **always** being scanned forward in the logical string sequence). The first value of the pair corresponds to the sequential position of the character in the input string in logical sequence. The second value of the pair corresponds to the weight assigned to the character according to level *m* in script <SPECIAL>.

In the ~~table~~Common Template Table, this behavior is described using the parameter couple "forward, position". To be conformant to this, the parameter couple "backward, position" shall never be specified for level *m* (see clause 5.4 THERE IS NO CLAUSE 5.4!). These two parameters shall be considered mutually exclusive.

In the Common Template table, definitions of these characters for levels 1 to 3 are such that they are ignored at these levels and values are exclusively assigned to level *m* (*m* being equal to 4 in the Common Template).

6.3 Common Template Table: formation and interpretation

This ~~section~~ clause specifies:

- the syntax used to form the Common Template Table in ~~Annex~~ annex A of this International Standard or a ~~table~~ tailored ~~starting from~~ table based upon the Common Template Table
- conditions of well-formedness of a table using this syntax
- interpretation of 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~~6.3.1 BNF syntax rules

Definitions between {curly brackets} make use of terms not defined in this BNF syntax, and assume general English usage.

Other conventions:

* means 0 or more repetitions of a token,
parentheses indicate optional occurrence of a token.

NOTE: THE USE OF SMART QUOTES HERE IS PECULIAR.

```
1.      character ::= {any member of the repertoire of the encoded
        character set in use}
2.      line_delimiter ::= {end-of-line in the text conventions in use}
3.      digit ::= '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
4.      hexdigit ::= digit|'A'|'B'|'C'|'D'|'E'|'F'|digit
5.      id_start ::=
        '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'
6.      id_part ::= id_start|digit|'-'|'_'|'+'
7.      comment_char ::= '%'
8.      space ::= ' '
9.      four_digit_hex_string ::= hexdigit hexdigit hexdigit hexdigit
10.     comment ::= comment_char character*
11.     identifier ::= id_start id_part*
12.     simple_symbol ::= '<' identifier '>'
13.     ucs_symbol ::= '<U' four_digit_hex_string '>'
14.     symbol ::= simple_symbol | ucs_symbol
15.     symbol_group ::= symbol |'"' symbol symbol* '"'
16.     level_token ::= symbol_group | 'IGNORE'
17.     delimited_level_token ::= level_token ';'
18.     multiple_level_token ::= delimited_level_token* level_token
19.     line_completion ::= ((space)comment) line_delimiter
20.     symbol_list_item ::= symbol
21.     symbol_list_item_range ::= symbol_list_item '..' symbol_list_item
22.     symbol_list_element ::= symbol_list_item_range | symbol_list_item
23.     symbol_definition ::= symbol_list_element
```

```
24.     symbol_weight_entry ::= symbol_list_item space multiple_level_token
        line_completion
25.     delimited_symbol_list_element ::= symbol_list_element ';'
26.     symbol_list ::= delimited_symbol_list_element* symbol_list_element
27.     section_identifier ::= identifier
28.     section_definition_simple_entry ::= 'section' space
        section_identifier line_completion
29.     section_definition_list_entry ::= 'section' space
        section_identifier space symbol_list line_completion
30.     section_definition_entry ::= section_definition_simple_entry |
        section_definition_list_entry
31.     target_symbol ::= symbol
32.     reorder_after_entry ::= 'reorder-after' space target_symbol
        line_completion
33.     reorder_end_entry ::= 'reorder-end' line_completion
34.     reorder_section_after_entry ::= 'reorder-section-after' space
        section_identifier space target_symbol line_completion
35.     direction ::= 'forward' | 'backward'
36.     delimited_direction ::= direction ';'
37.     multiple_level_direction ::= delimited_direction* direction
38.     order_start_entry ::= 'order_start' space identifier ';'
        multiple_level_direction space (',position') line_completion
39.     order_end_entry ::= 'order_end' line_completion
40.     simple_line ::= (symbol_definition | symbol_weight_entry)
        line_completion
41.     tailoring_line ::= (section_definition_entry | reorder_after_entry
        | reorder_end_entry | reorder_section_after_entry |
        order_start_entry | order_end_entry) line_completion
42.     table_line ::= simple_line | tailoring_line
43.     section ::= {ordered set of simple_line's –See I1 below.}
44.     untailored_template_table ::= simple_line*
45.     tailored_table ::= table_line*
46.     weight_table ::= untailored_template_table | tailored_table
```

~~6.3.2 Well-formedness Conditions~~6.3.2 Well-formedness conditions

WF1. Any *simple_symbol* occurring in a *multiple_level_token* must occur in a *symbol_definition* in the same *symbol_weight_entry* ~~that~~ in which the *multiple_level_token* occurs ~~in~~, or in a *symbol_weight_entry* that occurs earlier in the sequence of *table_line*'s that constitute a *tailored_table*. [~~I.e.~~That is, all *simple_symbol*~~'~~s must be "defined" before they are "used". Note that *hex_symbol's* are all assumed to be predefined.]

WF2. All *multiple_level_token*~~'~~s in a *tailored_table* must contain the same number of *delimited_level_token*~~'~~s. [That is~~I.e.~~, a tailorable table must be consistent in its use of levels throughout.]

WF3. A *tailored_table* may not contain a *multiple_level_direction* if it does not also contain a *multiple_level_token*. [That is~~I.e.~~, no *order_start statement* can be used in a table which defines no multi-level weights.]

WF4. A *multiple_level_direction* in a *tailored_table* must contain the same number of *delimited_direction*~~'~~s as the number of *delimited_level_token*~~'~~s of any *multiple_level_token* in that *tailored_table*. [That is~~I.e.~~, any *order_start* must have the same number of levels as is generally used in the table.]

WF5. If a *level_token* in a *multiple_level_token* consists of a *symbol_group*, all successive *level_token*'s in that *multiple_level_token* must also consist of a *symbol_group*. [That is~~i.e.~~, ~~don't use~~ 'IGNORE' may not be used at a level after an explicit symbol for a weighting.]

WF6. Any *section_identifier* occurring in a *reorder_section_after_entry* must occur in a *section_definition_entry* ~~that~~ which occurs earlier in the sequence of *table_line*'s that constitutes a *tailored_table*. [That is~~i.e.~~, all *section_identifier*'s must be "defined" before they are "used".]

WF7. No two *section_definition_entry*'s in a *tailored_table* may contain the same values in their *section_identifier*'s. [That is~~i.e.~~, multiple definition of *section*'s is prohibited; *section_identifier*'s must be unique.]

WF8. Each *reorder_after_entry* in a *tailored_table* must be followed by a *reorder_end_entry* or another *reorder_after_entry*.

WF9. If a *tailored_table* contains one or more *order_start_entry*'s, it must be terminated with an *order_end_entry*.

WF10. No *reorder_section_after_entry* may contain a *target_symbol* whose value is the same as any *symbol* in the *section_definition_list_entry* whose *section_identifier* is the same as the *section_identifier* in that *reorder_section_after_entry*. [That is~~i.e.~~, a *section* cannot be reordered after a line which the *section* itself contains; ~~prohibit~~ attempts at recursive relocation of lines are prohibited.]

~~6.3.3 Interpretation of Tailored Tables~~6.3.3 Interpretation of tailored tables

I1. 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_entry* ~~OR consists~~, or 2) of the list of *simple_line*'s following a *section_definition_simple_entry* in a *tailored_table*. [That is~~i.e.~~, a *section* is defined 1) by a specific *symbol_list*, or ~~just~~ 2) by taking all the lines following the *section_definition_entry* until ~~you hit~~ another tailoring line such as an *order_start_entry*, a *reorder_section_after_entry*, another *section_definition_entry*, or the end of the entire table, is encountered.]

I2. A *tailored_table* containing a *reorder_after_entry* is equivalent to the same *tailored_table* with the *table_line*'s between that *reorder_after_entry* and the first subsequent *reorder_end_entry* reordered 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_after_entry*, and with that *reorder_after_entry* and that *reorder_end_entry* removed. [That is~~i.e.~~, move the block of lines between the *reorder_after_entry* and the *reorder_end_entry* to follow the *target_symbol* and remove the *reorder_after_entry* and *reorder_end_entry* themselves.]

I3. A *tailored_table* containing a *section_reorder_after_entry* is equivalent to the same *tailored_table* with the *section* associated with that *section-reorder_after_entry* 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 *section_reorder_after_entry*, and with that *section_reorder_after_entry* removed.

I4. A *weight_table* is said to be in normal form when it contains no *reorder_after_entry*'s or *section_reorder_after_entry*'s. [In general, a *tailored_table* can be put into normal form by the operations implied by I1 and I2.]

I5. A *weight_table* in normal form is said to be evaluated when each *symbol_definition* in the *weight_table* is mapped to a positive integral numeric value such that those values ~~are~~ increase

monotonically [WHAT DOES THIS MEAN???] ~~increasing~~ by the order ~~that~~ in which the *symbol_definition*~~'~~s occur in the *weight_table*. (The *table_line*~~'~~s of the *weight_table* are first mapped to the set of positive integers, by sequential order in the file. This mapping defines an ordered set of line numbers. The *symbol_definition*~~'~~s are then mapped to a set of positive integers that varies monotonically with the set of line numbers.) [Note that this does not restrict the starting number for the weight of the first *symbol_definition*, nor does it require that the numbers for these weights be immediately consecutive.]

I6. An evaluated *weight_table* is said to be collation-element-weighted when each *simple_symbol* occurring in each *multiple_level_token* in that evaluated *weight_table* has been mapped to the integral value which corresponds to the *symbol_definition* ~~that~~ which contains the same *simple_symbol*. [I.e., each *multiple_level_token* can ~~now~~ be interpreted as containing either *symbol'*s mapped to integral weight values ~~or~~ , or as instances of ~~the string~~ 'IGNORE'. All *hex_symbol'*s are assumed to map to an integral weight value equal to that *hex_symbol* interpreted as a hexadecimal number. At this point the mathematical injection of strings can be defined using the *weight_table*.]

6.3.4 Conditions for considering specific table equivalences

C1 Two collation weighting tables W1 and W2 are said to be equivalent if, for all strings S defined on a repertoire R, any comparison of any two of the strings S1 and S2 based on their corresponding numbers from the injection P1 produced on W1 is matched exactly ~~matched~~ by the comparison of S1 and S2 based on their corresponding numbers from the injection P2 produced on W2. [That is~~I.e.~~, if one takes two strings, builds keys for each based on table 1 and compares them, one should always get the same results as when one builds keys for them based on table 2 and compares them.]

6.3.5 Conditions for results to be considered equivalent

C2. An implementation of international string ordering is conformant with this ~~standard~~ 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 ~~Section~~ clause 6.1.2 of this ~~standard~~International Standard.

C3. An implementation of international string ordering is conformant with this ~~standard~~ International Standard if any tailoring it uses can be demonstrated to be equivalent to a *weight_table* constructed according to the rules of clause 6.1.2 of this standard.

6.4 Declaration of a delta

It is recommended that tailoring be ~~done starting with~~ based upon the Common Template ~~table~~ Table described in annex A. If the format used in the Common Template Table is used, then tailoring can be accomplished using, for example, ISO/IEC 14652, which uses a syntax ~~that is~~ compatible with the one described in this International Standard.

Any declaration of conformance to this International Standard shall be accompanied with a declaration of the differences ~~that~~ occuring in the comparison table used relative~~ly~~ to the Common Template Table ~~if~~ when a fixed table is used by the application. These differences are called a *delta* according to this International Standard. Such a delta shall contain the equivalent of:

1. At least one valid *order_start_entry* described in clause 6.3.1; ~~many~~ an unlimited number of blocks containing an *order_start_entry* and an *order_end_entry* may be declared. The direction values may vary between “forward” and “backward” depending on ~~writing systems~~linguistic requirements.

2. The number of levels used for comparison.

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

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

In cases where ~~the~~ an application~~s~~ has provision to allow the end-user to tailor the table himself or herself, ~~any~~ a statement of conformance shall indicate which ~~ones~~ of the 4 elements of the previous list are tailorable and which ~~ones~~ 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*~~*NOTE: The declaration may use a different syntax from the one ~~proposed~~ 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+007A 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 < å < ä"

These two notations can reasonably be considered to be equivalent to the more precise expressions (which also give weights at levels 2 and 3):

```
reorder_after <U007A>
<U00E5> <U00E5>;<BLANK>;<MIN>
reorder_end

reorder_after <U00E5>
<U00E4> <U00E4>;<BLANK>;<MIN>
reorder_end
```

6.5 Name of the Common Template Table and name declaration

Whenever the Common Template Table is referred externally as a ~~starting~~ base point in a given context, ~~either applicative or contractual~~whether in an application, contract, or procurement requirement, it shall be referenced using the name ISO14651_1999_TABLE1. If another name is used due to ~~different~~ practical constraints, ~~any~~ a declaration of conformance shall indicate how the correspondence between this other name and the name ISO14651_1999_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, as that development will necessitate amendments to (at least) annex A of this International Standard.

Annex A -- Common Template Table (normative)

In this ordering table ~~constituting~~ , which constitutes a ~~common~~ Common ~~template~~Template Table, a number of characters and scripts of the world ~~are~~ is missing, due to the ~~non-inclusion of~~ fact that those characters or scripts ~~in the current stage of development of~~ have not yet been encoded in the reference character set repertoire, ~~that of~~ ISO/IEC 10646-1 (Universal multiple-octet coded Character Set, or UCS) at time of ~~publishing~~the preparation of this International Standard.

It is the intent of ISO/IEC to ~~complete~~ include the ordering of those scripts explicitly in the ~~common~~ Common ~~template~~ Template Table when~~ever~~ data becomes available, by way of amendments to this International Standard. If the ~~common~~ Common ~~template~~ Template Table is not tailored for unspecified characters, then an implicit order is assigned in the following table, which ~~might~~ may not meet the user requirements of a particular community. Any delta with this table shall be declared in ~~any~~ a statement of conformance to this International Standard as per the ~~prescriptions~~ specifications of the conformance clause.

Name used for referring to this table in this version of this International standard: **ISO14651_1999_TABLE1**

~~Note~~NOTE: The complete table can be found at URL

http://www.dkuug.dk/jtc1/sc22/open/n2844/n2844t1.txt

for the ~~whole~~ duration of the FCD ballot stage. When the DIS ~~will be~~ is produced, the complete table will be inserted in this International Standard. This table is intended to be machine-readable and is normally produced on paper only for checking or for reference purposes or for helping in ~~declaring~~ to declare a delta.

Brief excerpt of the table, which requires 520K of storage in plain text:

```
% escape_char /
% comment_char %

% LC_COLLATE

% Decomment the lines above to create a 14652-style
%   LC_COLLATE definition.

% Autogenerated LC_COLLATE weight symbol table
%   created from unidata.txt
% Equivalent to weights of basekeys.txt + compkeys.txt

% Order of internal symbols

<RES-1>
<BLK>
<MIN>
.
.
.
<BODKA>
<CJKVS>
<S0200>..<S1105> % 0x0200..0x1105
```

```
% order_start AllScripts;forward;forward;forward;forward,position

% Decomment the order_start line (and corresponding order_end line at
%   the end of this table) to specify directions for each level.
%   To tailor for French accent handling, or not to make French
%   a special case add an order_start statement
%   and order_end for Latin in the Latin section, as follows:

% order_start Latin;forward;backward;forward;forward,position

% <Uxxxx> <Base>;<Accent>;<Case>;<Special>

% Note that <Special> must be evaluated as exact hex value
%   and not as an autoweighted symbol.

<U0000> IGNORE;IGNORE;IGNORE;<@0000> % NULL
.
.
.
```

Annex B -- Benchmarks (informative)

B.1 Example 1 – Canadian delta and benchmark

~~The next few pages contain~~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 ~~the~~ illustration here; a ~~bigger~~ larger delta is required, mainly for special characters, for full conformance to this Canadian standard, ~~referenced~~ and is given here as an example only. The example's specifications are to be performed using the Common Template Table of annex A, with the following delta:

5. Block properties: only one block with the following properties:

```
order_start TABLE;forward;backward;forward;forward,position
```

6. Number of levels unchanged to 4.

7. No symbol change.

8. No other insertion, deletion or redefinition than:

   - æsorted as if it were separate letters "ae" at level 1. The letters "ae" are distinguished ~~only~~ at level 2 from the ~~joined digraph~~ letter "æ" and are ~~then~~ sorted before it. Upper case is distinguished from lower case at level 3.

   - ð sorted as if it were the letter "d" at level 1. ~~Letter~~ The letter "ð" is distinguished at level 2 from the letter "d" and is sorted before it. Upper case is distinguished from lower case at level 3.

   - þ sorted as if it were separate letters "th" at level 1. The letters "th" are distinguished ~~only~~ at level 2 from the letter "þ" and are ~~then~~ sorted before it. Upper case is distinguished from lower case at level 3.

~~*Note: the last two letters are not used in the benchmark but are part of the Canadian delta.*~~

Alternate formal ISO/IEC 14652 tailoring equivalent

```
copy ISO14651_1999_TABLE1
order_start TABLE;forward;backward;forward;forward,position
reorder-after <U00C6>
<U00E6>  <S6CD><S72D>;<COMPAT><COMPAT>;<MIN><MIN>;IGNORE   % <ae>
<U00C6>  <S6CD><S72D>;<COMPAT><COMPAT>;<CAP><CAP>;IGNORE   % <AE>
reorder-after <U1E0E>
<U00F0>  <S705>;<COMPAT>;<MIN>;IGNORE                      % <d->
<U00D0>  <S705>;<COMPAT>;<CAP>";IGNORE                     % <D->
reorder-after <U2122>
<U00FE>  <S88B><S781>;<COMPAT><COMPAT>;<MIN><MIN>;IGNORE   % <th>
<U00DE>  <S88B><S781>;<COMPAT><COMPAT>;<CAP><CAP>;IGNORE   % <TH>
reorder-end
```

1 Unordered list (required test as per Canadian standard CAN/CSA Z243.4.1-1998)

ou	orvar›ur	CÔTE
lésé	Canon	COTE
péché	lame	côté
vice-président	Bohême	coté
9999	0000	aide
OÙ	relève	air
haï e	gène	vice-president
coop	casanier	modelé
caennais	élevé	Thorvardur
lèse	COTÉ	MODÈLE
dû	relevé	maçon
air@@@	Grossist	MÂCON
côlon	vice-presidents' offices	pèche
bohème	Copenhagen	pêché
gêné	côte	medal
meðal	McArthur	ovoï de
lamé	Mc Mahon	pechère
pêche	Aalborg	ode
LÈS	Größe	péchère
vice versa	vice-president's offices	œil
C.A.F.	cølibat	
orsmörk	PÉCHÉ	
cæsium	COOP	
resumé	@@@air	
Bohémien	VICE-VERSA	
co-op	gêne	
pêcher	CO-OP	
les	révélé	
CÔTÉ	révèle	
résumé	çà et là	
Ålborg	MacArthur	
cañon	Noël	
du	île	
haie	aï eul	
pécher	Île d'Orléans	
Mc Arthur	nôtre	
cote	notre	
colon	août	
l'âme	NOËL	
resume	@@@@@	
élève	L'Haÿ-les-Roses	

2 List with required results as per Canadian standard CAN/CSA Z243.4.1-1998

@@@@@
0000
9999
Aalborg
aide
aï eul
air
@@@air
air@@@
Ålborg
août
bohème
Bohême
Bohémien
caennais
cæsium
çà et là
C.A.F.
Canon
cañon
casanier
cølibat
colon
côlon
coop
co-op
COOP
CO-OP
Copenhagen
cote
COTE
côte
CÔTE
coté
COTÉ
côté
CÔTÉ
du
dû
élève
élevé
gène
gêne
gêné
Größe
Grossist
haie
haï e
île
Île d'Orléans
lame
l'âme
lamé
les
LÈS
lèse
lésé
L'Haÿ-les-Roses
MacArthur
MÂCON
maçon
medal
me›al
McArthur
Mc Arthur
Mc Mahon
MODÈLE
modelé
Noël
NOËL
notre
nôtre
ode
œil
ou
OÙ
ovoï de
pèche
pêche
péché
PÉCHÉ
pêché
pécher
pêcher
pechère
péchère
relève
relevé
resume
resumé
résumé
révèle
révélé
orsmörk
Thorvardur
orvar›ur
vice-president
vice-président
vice-president's offices
vice-presidents' offices
vice versa
VICE-VERSA

B.2 Example 2 – Danish delta and benchmark

The following is a Danish delta following ISO/IEC 14652 tailoring with the assumption that character mnemonics ~~are~~ to be resolved into UCS identifiers to ~~fit~~ accord with the ISO14651_1999_TABLE1 template used in this International Standard (this formal specification corresponds to Danish standard DS 377 and to "Retskrivningsordbogen", the Danish orthography specification):

```
escape_char /
comment_char %

% The ordering algorithm is in accordance
% with Danish Standard DS 377 (1980)
% and the Danish Orthography Dictionary
% (Retskrivningsordbogen, 1986).
% It is also in accordance with
% Greenlandic orthography.

LC_COLLATE
collating-element <A-A> from "<A><A>"
collating-element <A-a> from "<A><a>"
collating-element <a-A> from "<a><A>"
collating-element <a-a> from "<a><a>"
copy ISO14651_1999_TABLE1
reorder-after <CAP>
<CAP>
<CAPITAL-SMALL>
<SMALL-CAPITAL>
<MIN>
reorder-after <SP>
<SP>    <SP>;<SP>;IGNORE;IGNORE
<->     <SP>;<->;IGNORE;IGNORE
<//>    <SP>;<//>;IGNORE;IGNORE
reorder-after <U24C6>
<kk>    <Q>;<COMPAT>;<MIN>;IGNORE

NOTE: <K><'> MUST BE INSERTED HERE AS EQUIVALENT TO CAPITAL Q IN ORDER TO CONFORM TO
GREENLANDIC PRACTICE; THIS MAY IMPLY THAT K' IN NON-GREENLANDIC CONTEXTS WILL BE SORTED
INCORRECTLY FOR DANISH. (ARGUMENT FOR ADDING CAPITAL KRA TO THE UCS?)

reorder-after <U2122>
<TH>    "<T><H>";"<TH><TH>";"<CAP><CAP>";IGNORE
<th>    "<T><H>";"<TH><TH>";"<MIN><MIN>";IGNORE
reorder-after <U1EF4>
% <U:> and <U"> are treated as <Y> in Danish
<U:>    <Y>;<U:>;<CAP>;IGNORE
<u:>    <Y>;<U:>;<MIN>;IGNORE
<U">    <Y>;<U">;<CAP>;IGNORE
<u">    <Y>;<U">;<MIN>;IGNORE
reorder-after <U1E94>
% <AE> is a separate letter in Danish
<AE>    <AE>;<BLK>;<CAP>;IGNORE
<ae>    <AE>;<BLK>;<MIN>;IGNORE
```

```
<AE'>  <AE>;<AIGUT>;<CAP>;IGNORE
<ae'>  <AE>;<AIGUT>;<MIN>;IGNORE
<A3>   <AE>;<MACRON>;<CAP>;IGNORE
<a3>   <AE>;<MACRON>;<MIN>;IGNORE
<A:>   <AE>;<COMPAT>;<CAP>;IGNORE
<a:>   <AE>;<COMPAT>;<MIN>;IGNORE
% <O//> is a separate letter in Danish
<O//>  <O//>;<BLK>;<CAP>;IGNORE
<o//>  <O//>;<BLK>;<MIN>;IGNORE
<O//'> <O//>;<AIGUT>;<CAP>;IGNORE
<o//'> <O//>;<AIGUT>;<MIN>;IGNORE
<O:>   <O//>;<TREMA>;<CAP>;IGNORE
<o:>   <O//>;<TREMA>;<MIN>;IGNORE
<O">   <O//>;<2AIGU>;<CAP>;IGNORE
<o">   <O//>;<2AIGU>;<MIN>;IGNORE
% <AA> is a separate letter in Danish
<AA>   <AA>;<BLK>;<CAP>;IGNORE
<aa>   <AA>;<BLK>;<MIN>;IGNORE
<A-A>  <AA>;<A-A>;<CAP>;IGNORE
<A-a>  <AA>;<A-A>;<CAPITAL-SMALL>;IGNORE
<a-A>  <AA>;<A-A>;<SMALL-CAPITAL>;IGNORE
<a-a>  <AA>;<A-A>;<MIN>;IGNORE
<AA'>  <AA>;<AA'>;<CAP>;IGNORE
<aa'>  <AA>;<AA'>;<MIN>;IGNORE
reorder-end
END LC_COLLATE
```

Benchmark 2 for Danish

A/S
ANDRE
ANDRÉ
ANDREAS
AS
CA
ÇA
CB
ÇC
DA
ÐA
DB
ÐC
DSB
D.S.B.
DSC
EKSTRA-ARBEJDE
EKSTRABUD
HØST
HAAG
HÅNDBOG
HAANDVÆRKSBANKEN
Karl
karl
NIELS-JØRGEN
NIELS JØRGEN
NIELSEN
oqararaaq
oKararaoK IS THIS RIGHT?
pequtai
peKutai
K'ânâK IS THIS RIGHT?
Qasigiangguit
Qeqertarsuaq
Qaanaaq IT'S NOT AN Å
RÉE, A
REE, B
Rée, B
RÉE, B
RÉE, L
REE, V
SCHYTT, B
SCHYTT, H
SCHÜTT, H
SCHYTT, L
SCHÜTT, M
SS
ß
SSA
STORE VILDMOSE
STOREKÆR
STORM PETERSEN
STORMLY
 orsmörk
THORVALD
THORVARDUR
ÞORVARÐUR
THYGESEN
VESTERGÅRD, A
VESTERGAARD, A
VESTERGÅRD, B
ÆBLE
ÄBLE
ØBERG
ÖBERG

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. Examples are:

- ~~duplicating~~ duplication of a string such as "41" for phonetic ordering into ~~3~~ differentiated strings for ~~trilingual~~ multilingual phonetic ordering usage (Irish Gaelic, German, English, and French ~~, English and German~~"):

  DAICHEAD A hAON

  ~~QUARANTE-ET-UN~~

  EINUNDVIERZIG

  ~~FORTY-ONE~~

  FORTY-ONE

  QUARANTE-ET-UN

  ~~EINUNDVIERZIG~~

- ~~removing~~ removal or ~~rotating~~ rotation of characters that are a nuisance for special requirements of ordering; for example, in France, removing "de" in "de Gaulle" and not removing "De" in "De Gaulle" (the former indicating ~~according to~~ noble origin, the latter ~~or~~ not), to give:

  Gaulle (de)

  De Gaulle

- transformation of ~~incomplete~~ abbreviated data into a fuller form; for example, transformation of "Mc Arthur" to give "Mac-Arthur"

- transformation of numbers so that the result will be ordered in numerical order ~~and not positionally~~, as opposed to positional order (see ~~specific section presented hereafter~~below). Numeric ordering is particularly delicate and requires special consideration in many ~~specific~~ cases.

C.2 Handling of numeral substrings in collation

A numeral is a string representing a number. ~~We will~~ The examples here ~~only~~ deal with numerals ~~that~~ which represent values in *R*, the real numbers, or subsets of *R*, as these have a ~~total~~ predetermined order. ~~We will also o~~Only ~~be dealing with~~ decimal numerals are dealt with in the examples given here.

The same principles apply to, for example, hexadecimal numerals, with the caveat that there are some words that look like hexadecimal numerals (such as English AD, BE, ABED, BEDE, CEDE, DEAD, DEAF,

DEED, FACE, FADE, FEED), and one must be careful to distinguish which are words and which are hexadecimal numerals. In some cases one uses ~~run-together~~ numerals run together, ~~perhaps~~ which may also be mixed with other substrings. ~~-~~This may happen for instance for part numbers, some date formats, and the like, where it is not obvious which substrings are really separate numerals. Run-together numerals ~~will~~ are not ~~be~~ discussed below.

The presentation below will ~~start with~~ 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 which sometimes uses another syntax with digits for numerals (~~like~~ such as Hàn numerals), and Roman numerals.

C.2.1 Handling of 'ordinary' numerals for natural numbers

The Common Template Table has no means of sorting 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 Common Template Table this International Standard ~~gives~~ yields the following list of "sorted" strings:

Release 1
Release 12
Release 2
Release 20
Release 9

(It is sufficient ~~to~~ simply to look positionally ~~look~~ at just the first digit in each numeral to see why ~~one gets this order~~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 ~~numbers~~ ordering is to ~~zero-~~pad them with zeroes to a given number of digits. If one ~~zero-~~pads ~~up to three digits~~ the numerals in our original example strings up to three digits, the following will result~~one gets~~:

Release 001
Release 020
Release 012
Release 002
Release 009

Using the Common Template Table defined in this International Standard one then ~~gets~~ 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:

1. One ~~has to~~ 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 ~~larger~~ greater that there are actually occurring numerals with more digits than one ~~pads~~ has padded up to, which results in partially getting back to the original situation, where the numerals'~~s~~ values are not taken entirely into ~~(full)~~ account.

2. Determinacy is lost, if some of the original numerals were already partially zero-padded. ~~E.g.~~ 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 ~~with~~ applying the same rules. Indeterminacy in ~~the~~ collation is not desirable.

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 ~~that~~ 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). ~~E.g.~~For example, given the original strings:

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

One ~~get~~ obtains after this preparation the following strings:

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

Which would be collated by the basic mechanism of this ~~standard~~ International Standard to:

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

As normally presented to the user:

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. ~~E.g.~~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 collated 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

As normally presented to the user:

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

The originally zero-padded numerals consistently come~~s~~ 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 ~~after~~ following the numerals.

~~There being~~ 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 ~~standard~~International Standard.

C.2.2 Handling of positional numerals in other scripts

ISO/IEC 10646 ~~has~~ 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~~, but r~~; rather ~~that~~, one should consider each maximal substring of digits of the same script ~~are each considered~~ to be a numeral.

C.2.3 Handling of other non-pure positional system numerals or non-positional system numerals (e.g. Roman numerals)

Chinese and ~~a few~~ some other ~~scripts~~ languages can use decimal digits (in the Hàn script ~~for Chinese~~, 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 ~~copying~~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 ~~the~~ 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 “Lou is 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). This generally requires context to be resolved with total certainty.

C.2.4 Handling of numerals for whole numbers

If negative whole numbers are also to be sorted according to their value, there are a number of issues to be considered. ~~-~~Most ~~often~~frequently, negative whole values are given numerals that begin with a negation sign. The negation sign ~~can~~ may be HYPHEN-MINUS U+002D (caveat: ~~it~~ this character may ~~be~~ represent a true hyphen, rather than a negation), or MINUS SIGN U+2212. But there are other conventions also, like using a ~~SLASH~~ SOLIDUS U+002F or a PERCENT SIGN U+0025 to indicate negativeness; or the negation ~~indication~~ 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 color (often red, which is not used in plain text and is therefore outside the scope of this International Standard). ~~We will i~~In the examples here, only ~~deal with~~ 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.

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):

3. 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 at the original positions. This step ~~is to~~ ensures determinacy.

4. ~~See to~~ Ensure that all of the copies have an explicit initial sign ~~indication~~indicator.

5. 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.

6. Between the sign ~~indication~~ 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).

7. 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*. ~~I.e.~~ That is, 9's complement of 0 is 9, of 9 is 0, of 5 is 4, etc.

8. Done with this (part of the) preparation.

For the basic collation step, use a tailoring of the template given in this standard~~. A~~, 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 ~~that~~ this is not a prerequisite for getting an acceptable ordering.)

Our example strings after this prehanding:

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

Sort these, using the basic mechanism of this standard:

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

As presented to the user:

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~~ which may have numerals for whole numbers in them (also if there are several such numerals in some of the strings), ~~that is~~ 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.2.5 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 ~~often~~ generally either ~~PERIOD~~FULL STOP U+002E, or COMMA U+002C. These characters also have other uses, also in conjunction with digits.

For the example, assume that ~~PERIOD~~ 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 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

After sorting:

–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.2.6 Handling of positive positional numerals with fraction parts and exponent parts

For very ~~big~~large, or very ~~tiny~~small, values, one often uses formats like $2.5*10^7$ (to ~~just pick~~ illustrate just one possible way of writing these for the purposes of the examples here). Here there is already an exponent [THIS BEGS THE QUESTION WHETHER PLAIN TEXT U+2077 IS USED OR RATHER U+0037 WITH FANCY TEXT SUPERSCRIPT FORMATTING], 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):

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

After sorting:

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

As presented to the user:

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

C.2.8 Handling of date and time of day indications

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

The preparation needed to ~~get~~ obtain date and time-of-day indications, of some predetermined syntaxes, sorted according to point in time is similar to what has been described above.

9. 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.

10. 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.

11. 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.

12. 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.

13. Use the UTC time zone for the date and time-of-day indications. Remove time zone indications, if present, in the date-time indication copies.

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

15. 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, ~~or...~~etc. No indeterminacy regarding year due to abbreviations like these may be present after the preparation step.

16. For years ~~A.D.~~CE, use an initial PLUS SIGN. For years ~~B.C.~~BCE, use an initial MINUS SIGN. Remove the original ~~A.D.~~CE or ~~B.C.~~BCE indication from the copies. (To ~~be~~ nitpick~~ing~~, year *n* B.C. should be represented by year (1–*n*), which is less or equal to zero if *n* is positive.)

17. 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.~~..~~

18. 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.

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

20. Alternatively, use a number indicating the point of time on a linear time scale (~~e.g.~~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).

21. Done with this (part of the) preparation.

22. 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 ~~B.C.~~BCE
Dated: Sept. 20, 1995, at 1 p.m. PST
Dated: 11-june/345 ~~A.D.~~CE [WHAT IS THIS FORMAT?]

After preparation:

Dated: +41955-07-19T13:00Z July 19, 1955, at 1 p.m. GMT
Dated: –780-01 January, 20 ~~B.C.~~BCE
Dated: +1995-09-20T10:00Z Sept. 20, 1995, at 1 p.m. PST
Dated: +3345-06-11 11-june/345 ~~A.D.~~CE

After sorting:

Dated: –780-01 ~~.~~January, 20 ~~B.C.~~BCE
Dated: +3345-06-11 11-june/345 ~~A.D.~~CE
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 ~~B.C.~~BCE
Dated: 11-june/345 ~~A.D.~~CE
Dated: July 19, 1955, at 1 p.m. GMT
Dated: Sept. 20, 1995, at 1 p.m. PST

C.2.9 Making numbers less significant than letters

In many cases numerals preceding letters should be considered as less significant than the following alphabetic part. But the Common Template Table ~~has~~ specifies digits ~~as~~ 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 the section C.2.10).

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)~~. C~~; chemical compound names which have prepended numerals, e.g., 1,2-diclorobenzol.

C.2.10 Maintaining determinacy

As noted above in several cases, ~~we have duplicated~~ 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 ~~duplicating~~ duplication for 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 ~~do~~ perform the preparation (without additional duplication) on the first half of the “doubled” string.

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. Th~~e present s~~is 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, ~~though~~however.

Note that the string after preparation is used only for the collation key construction. The original string is not intended to be retrievable ~~form~~ from the modified string, though ~~that~~ this is possible with this way of attaining determinacy. The strings to be presented to the user are the original, by preparation untouched, strings.

~~To maintain~~ Maintenance of determinacy when some of the original strings to be collated are identical, is out of the scope ~~for~~ of this ~~standard~~International Standard. A collation processor should, however, document if it is ‘stable’ (maintaining initial relative order of identical strings) or not. This is useful to know when collating on one field of multi-field data.

C.3 Posthandling

In case of equality established according to this International Standard on two character strings, it may be necessary to establish *a posteriori* an ultimate differentiation based on the original record being processed which may contain additional data. Although this posthandling is not part of the scope of this International Standard, consideration is given here on an informative basis to this ultimate stage of ordering.

Posthandling is necessary for modifying a resulting ordering key, or appending the original character string to an ordering key so that the results of comparisons can determine differences particularly in cases where homography results from the preparation phase. For example, there could be equivalencies if numerical values (for example, "010" and "10") may have been rendered identical in the preparation phase. A strict implementation of this International Standard ~~has no knowledge~~ will not recognize that the original strings are different in such cases, but the predictability requirement may still exist and posthandling will then be required to achieve this specific requirement.

Another case in point exists, for example, where different coding methods have been used in the original strings to be ordered in the same process. An optional posthandling phase can then determine internal differences even when results would appear exactly the same on paper for end-users (for example, an ISO/IEC 2022 input stream intermixing ISO/IEC 6937 and ISO/IEC 8859-2 coded characters). This may be required for internal processing of applications and maintaining integrity of comparison between records and even entire files.

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

Why aren't existing standard codes, character by character comparisons, and commercial sort programs appropriate for sorting, and what must be done to solve the problem? For clarity, this discussion will start with the Latin script.

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

Ex.Example: Sorting 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 doneperformed:

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

which This ordering is obviously wrongincorrect.

ii. Translating Transforming lower case to upper case and removing special characters gives yields a sorted list acceptable to users, but also yields unpredictable results.

Ex.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", "co-op", "coop" may give a different order with this method:

august
August
co-op
coop

iii. If accented characters are introduced using for example ISO/IEC 8859-1 code, the same problems encountered in steps examples i and ii above are amplified but they share the same causes.

iv. If tables are reorganized to make all related characters contiguous, one might think ~~it would permit~~that a simplified single-character sort would result, but this does not work either. Take upper and lower case unaccented letters as an example. If code ~~point~~ position 01 is assigned to "a", code position ~~point~~ 02 assigned to "A", code position ~~point~~ 03 to "b", code position ~~point~~ 04 to "B" and so on, ~~let's see what happens in~~ a list sorted directly ~~by~~ 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 ~~also~~, but remains obviously ~~wrong in~~incorrect for any country ~~from a~~ with regard to cultural ~~point of view~~expectations.

v. The only ~~path of~~ solution is to decompose the initial data in a way ~~that~~ which will respect traditional lexical order, and at the same time ensure absolute predictability. For the Latin script, this necessitates at least four levels:

1. The first decomposition renders information to be sorted ~~case~~ case-insensitive and insensitive to diacritical ~~mark insensitive~~marks, ~~and removes~~removing all special characters (which have no pre-established order in any ~~human~~ [WHAT, THEY HAVE AN ORDER AMONG THE BEES AND ANTS?] culture):

An example using English:

> "résumé" (~~an English word derived from French but with a very different meaning in French~~'curriculum vitae') becomes "resume" ('begin again'), without any accent.

An example using French:

> "Vice-légation" becomes "vicelegation", with no accent, no upper case and no ~~dash~~hyphen.

An example using German:

> "groß" becomes "gross", with the sharp-s being converted to double-s to render it case insensitive.

In Spanish or ~~Scandinavian~~ Nordic languages, some extra letters are added to the 26 fixed letters of the English, French and German alphabets, which are not ordered according to the expectations of ~~this group of~~those languages. This ~~calls~~ demonstrates the need for adaptability.

2. The second decomposition breaks ties on quasi-homographs, that is, strings that differ only because they have different diacritical marks. In ~~the~~ English ~~example above~~, "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 ~~say~~ explicitly specify ~~if~~ whether "resumé" ~~(another spelling)~~ should come

before "résumé", ~~which~~ though this would seem logical: most English and German dictionaries only state that unaccented words precede the accented words (often nothing is stated but the practice can be determined from the order of the headwords).

Here another characteristic is introduced. In French, because of the large number of multiple quasi-homograph groups formed of more than 2 instances, ~~main~~ the most important dictionaries follow ~~a~~ the following rule ~~that is the following~~: 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 being then assigned to each type of accent. ~~For example~~According to this, "coté" should be sorted after "côte" but before "côté". This is easy to implement with "backwards" tailoring as described in clause 1.5 [CHECK REFERENCE]: a number is assigned to each character of ~~original~~ the data to be sorted, representing either a letter with an accent or a letter with no accent at all, but these numbers are stacked instead of being added to a linear list: in other words, the resulting string is made starting from the last character of the original data and backward. [I DON'T THINK THIS LAST SENTENCE MAKES SENSE]

Example: to obtain ~~the following~~ 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. Here this scheme is sufficient to break the tie correctly at this second level.

3. The third decomposition breaks ties for quasi-homographs ~~different~~ which differ only because upper-case and lower-case characters are used. This time, the tradition is well established in ~~English and~~ German dictionaries, where lower case always precedes upper case 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 lower case letters are mixed, the capitals generally come first, ~~but~~ though this is not an established and stated rule, because there are numerous exceptions. In some English dictionaries, such as the Concise Oxford, capitals precede smalls, but in others the reverse is done. In the ~~So for a~~ Common ~~Template it is advisable to use English and~~ the German ~~traditions~~tradition has been followed~~, if one wants to group the largest possible number of languages together. Let's note here by the way~~. Note that in Denmark, upper case ~~comes before~~ is specified to precede lower case, a different but ~~well~~ well-established rule. This is a second fact ~~calling~~ which demonstrates the need for adaptability in the model used in this ~~standard~~International Standard.

Example: to have the following order: "august", "August", numbers could be assigned indicating respectively "llllll", "ullllll", where "l" means lower case and "u" upper case. [THE ACTUAL SYNTAX SHOULD BE QUOTED HERE]

4. The fourth decomposition breaks the final tie ~~that~~ which, in general, does not correspond to any strong tradition, namely, the tie ~~due~~ between~~to~~ quasi-homographs ~~that differ~~differing only because they contain special characters. ~~-~~Breaking this tie is essential to ensure the absolute predictability of ~~sorts~~ ordering ~~and also to be able to sort~~ 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 orders of decomposition, simply putting them ~~in row~~ sequentially in the fourth order of decomposition would mean that their position would be lost. These positions are quite important to solve remaining ties and in consequence ~~we must retain here~~ the original positions of these special characters must be retained: two quasi-homographs could each contain a common special character in different positions and thus be strictly different (~~ex.:~~example: "ab*cd" is ~~still~~ different from "a*bcd" despite they share one and only one common special character).

Example: to ~~have~~ obtain the following order: "coop", "co-op", "coop-", numbers could be assigned respectively according to the following pattern: "d", "d3-" and "d5-", where "d" is an ~~always~~ever-present delimiter ~~that separates~~separating this decomposition from the first three in case all four decompositions are to be concatenated to form a single sorting key based on numeric values (see discussion in the next paragraph). "3-" means a ~~dash~~ hyphen in position 3 of the original string. "5-" means a ~~dash~~ hyphen in position 5, and so on.

These four decompositions can be structured using a four-level key, concatenating the subkeys from the highest significance to the lowest. If the coded assignment of numbers is done properly, instead of necessitating a cumbersome exception process for dealing with homographs, all decompositions may be made at once and resulting strings concatenated and passed through a standard ~~sort~~ ordering program sorting in numeric order. To attain this result, it is sufficient that the numbers chosen for the first decomposition code set be greater than numbers chosen for the second one, the second one's greater than the third one's, and that the delimiter chosen for the fourth decomposition be less than the lowest possible number coded elsewhere for the sort (a delimiter called logical zero), in which case no restriction applies to the content of the fourth decomposition. An easier implementation might just choose to put the lowest value possible as a delimiter between each subkey, in which case no restriction ever applies.

This method ~~has been~~was fully described with tables for the first time in *Règles du classement alphabétique en langue française et procédure informatisée pour le tri*, Alain LaBonté, Ministère des Communications du Québec, ~~19 août 1988~~1988-08-19, ISBN 2-550-19046-7.

Reduction techniques have been designed to considerably shorten space requirements. As no implementation is required to use specific numbers for weights and ~~does not require~~ neither reduction nor compression is required, this issue is outside the scope of this ~~standard~~ International Standard. Nevertheless, ~~but~~ it is interesting to note that implementation can be optimized. This has been improved over time and is ~~highly feasible~~easy to accomplish.

A public-domain reduction technique is described in details (with ~~ample~~ numerous examples) in *Technique de réduction - Tris informatiques à quatre clés,* Alain LaBonté, Ministère des Communications du Québec, ~~June~~ 1989-06 (ISBN 2-550-19965-0).

vi. For a number of languages, the Common Template presented in this standard will need to be adapted, both in the table values for the four orders of keys (which can require redefining characters or introducing multicharacter collating elements into the table) and 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 which native order is not in the Common Template table:

Traditional Spanish ~~(~~note where "ch" is greater than "cu" and "ña" is greater than "no"):

cuneo<cúneo<chapeo<nodo<ñaco

(Comparative French/English/German sort:

chapeo<cuneo<cúneo<ñaco<nodo)

Danish (~~note~~ 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<cødibat< Aachen<Aalborg<Århus

(Comparative French/English/German sort:
Aachen<Aalborg< Alzheimer<Århus<cæsium<cødibat<czar)

vii. It is important that in all coding environments, and in all programming environments, the order be consistent so that ~~sort~~ ordering programs ~~can give~~yield reliable results re-useable in programs; conversely, comparisons of two character strings where an order is expected should ~~be in line~~accord with results given by ~~sort~~ ordering programs. ~~Hence~~ Therefore, it is advisable that all processes which expect a given order ~~all~~ use the same comparison API. This ~~standard~~ International Standard has ~~built~~ been based on this requirement ~~that~~, which fomerly was not respected ~~before.~~.

Furthermore, it should be possible to have access, externally, to the ~~ultimate~~ binary strings on which real comparison is made. This will allow old processes which can not be changed easily but which are able to sort raw binary data, to sort in a consistent way with new processes. This ~~standard~~ International Standard allows this practice, while ~~it~~ also ~~provides~~ providing a way to avoid completely ~~avoid~~ the use of such binary strings.

Annex E -- BIBLIOGRAPHY

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

DS 377 (1980) – DS 377:1980 *Alfabetiseringsregler* – Dansk Standard

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 15: Latin alphabet No. 9*

ISO/IEC 14652, *Information Technology -- Specification Method for Cultural Conventions* (FCD)

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

END OF THIS INTERNATIONAL STANDARD