ISO/IEC JTC un/SCvingt-deux/WG20 N536

Date: 19quatre-vingt-six

quinze mille quatre cent trente-cinq

ISO/IEC JTC un/SCvingt-deux/WGvingt

Secretariat: **ANSI**

Information Technology — Internationalization APIs

Technologies d'information — *IPA concerant l'internationalisation*

Document type: **International Standard**
Document subtype: **Not applicable**
Document stage: **Error! Number cannot be represented in specified format.**
Document language: **E**

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

ISO/IEC 15435 was prepared by ISO/IEC JTC 1/SC22/WG20 Internationalization. No other international organization contributed to the preparation of this standard.

This standard does not cancel or replace other documents.

The standard provides interfaces to data as recorded with ISO/IEC 14651 and ISO/IEC 14652.

Annex B is normative.

Annex A is informative.

This is WD1 of the standard

Introduction

The document consists of an introductory section, a section on locale selection, a section on a set of APIs for transformation including character set conversion and transliteration of strings, a section on collation, a section on formatting of message strings, a section of cultural data formatting and a section on string handling

The typography has been made for easy distribution over networks with restricted layout capabilities, such as email, news and ftp that lacks possibilities for font information. Keywords etc are thus indicated in quotes.

Information Technology — Internationalization APIs

1 Scope

The purpose of this standard is to specify a set of APIs for internationalization. It contains a functional overview, and a specification of the individual APIs with the interface specification and a semantics specification. The functional description is done in a programming-language-independent manner using the ISO/IEC 13884 and ISO/IEC 11404 standards.

The APIs cover support for multiple coded character sets, including ISO/IEC 10646 support and transformations between coded character sets, functional support for the internationalization data specifiable by ISO/IEC 14652, and string handling.

2 Conformance

A conforming binding is a language binding that binds to all the APIs in this standrard and with semantics as specified in this standard. For APIs with more then one version with the same functionality (indicated by use of a letter in the identification) a conforming binding needs only to bind to one version of the API.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this International Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards

ISO/IEC 11404:1996, Information Technology - Language Independent Datatypes

ISO/IEC 13884:1996, Information Technology - Language-Independent Procedure Calling

ISO/IEC 10646-1:1996, Information Technology - Universal Multiple-Octet Coded Character Set (UCS) (incl AMD 1-4 and TCOR1)

ISO/IEC 9899:1996, Information Technology - Programming Language C (incl AMD 1 and TCOR 1)

ISO/IEC CD 14651, Information Technology - International default Sorting Specification

ISO/IEC CD 14652, Information Technology - Cultural conventions Data Specification Format

CEN ENV 12005:1996, Information Technology - Procedures for European Registration of Cultural Elements

4 Terms and definitions

For the purposes of this International Standard, the terms and definitions given in the following apply.

4.1 API - an Application Programming Interface, that describes the interface between the application software and the application platform for the service offered by the specification. The APIs are described in the form of a functional description of a function with its name and parameters and return values.

4.2 Transliteration - transformation of characters of one language into characters of another language.

NOTE Transliteration of the same script may be different, for example a Serbian text and a Russian text, both written in the Cyrillic script, may have different transcription rules for the language Danish. The transformation may be between different scripts, as in the previous example, or within the same script, for example Swedish to Danish where ö may become ø, and ä become æ.

4.3 Transcription - transformation of sounds of one language into characters of another language.

NOTE Transcription has the same characteristics as noted for transliteration.

5 String, encoding, repertoire, and locale data types

As basic string handling is dependent on the user's preferences (as given via the string), encoding, repertoire, and locale data types are described together here.

5.1 String data type

The string handling APIs defined in this International Standard operate on an internal representation of character strings, which are arrays of characters, with an associated lenght. The string type can hold all characters of ISO/IEC 10646 including null characters - type character (ISO(1) standard(0) 10646). The string data type shall be independent of the locale. All length information is given in terms of characters (not storage units). An empty string is indicated by the implementation-defined value NIL.

5.1.1 procedure newstring(in length: integer) returns string.

The "newstring" procedure creates a string object with the necessary space to hold "length" characters. The string is filled with null characters. The "newstring" procedure returns the string, if there is no memory available it returns the empty string.

5.1.2 procedure freestring(in s: string) returns integer.

The "freestring" procedure frees the memory occupied by the string "s". It returns 0 if the opertion is succesful, and 1 otherwise.

5.1.3 procedure stringlen(in s: string) returns integer.

The "stringlen" procedure returns the length of the string. If the string is empty it returns 0.

5.2 Encoding data type

The "encoding" data type holds data necessary to convert to and from an external encoding to the internal string representation. This includes mapping of coded characters to the internal repertoire, how to shift between subencodings such as via ISO 2022 techniques, or representation via symbolic character names identified via introducing sequences, and state information.

NOTE The encoding definition is closely related to the "charset" definition in the Internet MIME specification, POSIX localedef functionality and newer developments for the C and C++ programming languages.

5.2.1 procedure newencoding(in encodingname: string, out enc: encoding) returns integer.

The "newencoding" procedure creates an encoding object with the necessary space to hold all information necessary to convert between the encoding and the internal string representation. The "newencoding" procedure sets default values, including the "line_terminator" to being "CR""LF", the "invalid_char" to being "SUB", the "symbolic_char_introducer" to being "NUL" (not valid), the "sub_encoding_change" procedure, the "get_symbolic_char_name" procedure and the "put_symbolic_char_name" procedure to be the null procedure, and the "input_state" and the "output_state" variables to be the initial state .

The "encodingname" is an implementation defined string with the following characterisics:

An initial string of "std/" refers to the charmaps registered in the international cultural register, CEN ENV 12005.

If the specified encoding is valid and supported, the "newencoding" procedure allocates memory for the new object and returns a pointer to the object in "enc". It is the application's responsibility to free this memory with a call to the "freeencoding" procedure when the object is no longer needed. If the procedure fails for any reason, the contents of "enc" is undefined.

The "newencoding" procedure returns one of the following values:

LC_SUCCESS - The procedure call was successful

LC_NOTSUPPORTED - The encoding is not supported by the current system.

LC_NOMEMORY - there was insufficient memory to perform the procedure

LC_INVALID - The specified encoding is invalid

5.2.2 procedure freeencoding(in enc: encoding) returns integer.

The "freeencoding" procedure frees the memory occupied by the encoding "enc". It returns a zero if the operation is succesful, and a 1 otherwise.

5.2.3 procedure setencint(in enc: encoding, in param: string, in val: integer) returns integer.

The "setencint" procedure sets a specific parameter as specified in the string "param" of the encoding specification to a specific integer value as specified in "val". The defined parameters are:

(to be described)

It returns a zero if the operation is succesful, and a 1 otherwise.

5.2.4 procedure setencbytes(in enc: encoding, in param: string, in val: pointer to sequence of octet, in len: integer) returns integer.

The "setencbytes" procedure sets a specific parameter as specified in the string "param" of the encoding specification to a specific multibyte value as specified in "val" with the length "len" bytes. The defined parameters are:

"line_terminator"

"invalid_char"

"symbolic_char_introducer"

5.2.5 procedure setencproc(in enc: encoding, in param: string, in val: procedure) returns integer.

The "setencproc" procedure sets a specific parameter as specified in the string "param" of the encoding specification to a specific procedure value as specified in "val" . The defined parameters are:

"sub_encoding_change" - a procedure with no parameters (NOTE, is this true??) returns integer.

"get_symbolic_char_name" - procedure gscn(out c:character, inout p: pointer to sequence of octet, in len:integer) returns integer.

The application defined "get_symbolic_char_name" procedure is called by the "bytes2string" procedure when the character sequence in "symbolic_char_introducer" is met in the input octet sequence. It gets a pointer "p" to the first octet after the "symbolic_char_introducer" and determines whether there is a symbolic character according to the application functions definitions, with or without a terminator sequence, within "len" octets after the "p" pointer. If successful the procedure returns the found character in the internal string representation in "c" and the pointer "p" to the first octet after the symbolic character, including the possible terminator sequence. The application defined procedure returns:

0 if succesful

1 if it could not recognise a symbolic character within the "len" octets. "p" is not changed.

2 if the octet sequence is invalid according to the rules of the application. "p" is not changed.

"put_symbolic_char_name" - procedure pscn(in c: character, inout p: pointer to sequence of octet, in len: integer) returns integer.

The application defined "put_symbolic_char_name" procedure is called by the "string2bytes" procedure when a character is not present in the external encoding. It gets a pointer "p" to the next octet to be written in the sequence of octets and determines whether there is room to put a symbolic character according to the application functions definitions, with the "symbolic_char_introducer" value and with or without a terminator sequence, within "len" octets after the "p" pointer. If successful the procedure returns the pointer "p" to the first octet after the symbolic character written, including the possible terminator sequence. The application defined procedure returns:

0 if successful

1 if the procedure was not able to write the symbolic character within "len" octets. The pointer "p" is then left unchanged.

2 if the procedure had no means of writing the character "c", The pointer "p" is then left unchanged.

5.3 Repertoire data type

The "repertoire" data type holds data necessary for the "stringtrans" transliteration procedure.

5.3.1 procedure newrepertoire(in repertoirename: string, out rep: repertoire) returns integer.

The "newrepertoire" procedure creates a repertoire object with the necessary space to hold all information necessary The "repertoirename" is an implementation defined string with the following characterisics: An initial string of "std/" refers to repertoiremaps registered in the international cultural register, CEN ENV 12005. If the specified repertoire is valid and supported, the "newrepertoire" procedure allocates memory for the new object and returns a pointer to the object in "rep". It is the application’s responsibility to free this memory with a call to the "freerepertoire" procedure when the object is no longer needed. If the procedure fails for any reason, the contents of "rep" is undefined.

The "newrepertoire" procedure returns one of the following values:

0 - The procedure call was successful

1 - The repertoire is not supported by the current system.

2 - There was insufficient memory to perform the procedure

3 - The specified repertoire is invalid

5.3.2 procedure freerepertoire(in rep: repertoire) returns integer.

The "freerepertoire" procedure frees the memory occupied by the repertoire "rep". It returns 0 if the operation is succesful, and 1 otherwise.

5.3.3 procedure enc2repertoire(in enc: encoding, out rep: repertoire) returns integer.

The "enc2repertoire" procedure generates a repertoire object with a repertoire corresponding to the character repertoire of the encoding "enc". If the procedure is successful, it returns the repertoire object in "rep". It has the same return values as the "newrepertoire" procedure.

5.4 Locale data type

The "locale" data type is a pointer to a record with a number of variables capable of holding information sufficient to service all language-dependent internationalization services. The "locale" data type has provisions to affect groups of functionalities in categories, which are:

LC_COLLATE

LC_CTYPE

LC_MONETARY

LC_NUMERIC

LC_TIME

LC_MESSAGES

The category LC_ALL denotes all of the above categories.

The category NULL denoted a void category.

The "locale" data type includes the following variables (which are further described in ISO/IEC 14652):

LC_MONETARY values:

int_curr_symb: string.

currency_symbol: string.

mon_deccimal_point: string.

mon_thousands_sep: string.

mon_grouping: string

positive_sign: string.

negative_sign: string.

int_frac_digits: integer.

frac_digits: integer.

p_cs_precedes: integer

p_sep_by_space: integer.

n_cs_precedes: integer

n_sep_by_space: integer

p_sign_posn: integer

n_sign_posn: integer

LC_NUMERIC values:

decimal_point: string

thousands_sep: string

grouping: array of integers

LC_TIME values:

abday: array (1,7) of string

day: array (1,7) of string

abmon: array (1,12) of string

mon: array (1,12) of string

d_t_fmt: string

d_fmt: string

t_fmt: string

am_pm: string

t_fmt_ampm: string

era: string

era_year: string

era_d_fmt: string

alt_digits: array (1,100) of string

LC_MESSAGES values:

yesexpr: string

noexpr: string

5.4.1 procedure newlocale(in category: integer, in localename: string, out lc: pointer to record locale) returns integer.

The "newlocale" procedure creates a locale object with all the necessary information to perform the language-sensitive operations of internationalization procedures accepting an argument of the type "locale". If the procedure is successful, all categories of the locale object are created and initialized. Any categories in the locale identified by "localename" are initalized to the POSIX locale.

The "localename" is an implementation-defined string with the following characteristics:

- An initial string of "std/" refers to the locales registered in the international cultural register, CEN ENV 12005.

If the specified locale is valid and supported, the "newlocale" procedure allocates memory for the new object and returns a pointer to the object in "lc". It is the application's responsibility to free this memory with a call to the "freelocale" procedure when the object is no longer needed. If the procedure fails for any reason, the contents of "lc" is undefined.

The "newlocale" procedure returns one of the following values:

0 - LC_SUCCESS - The procedure call was successful.

1 - LC_INCOMPLETE - The specified locale has been created, but the locale object contains one or more categories that were initialized to the POSIX locale because the "localename" did not identify a value for that category.

2 - LC_NOTSUPPORTED - The locale is not supported by the current system.

3 - LC_NOMEMORY - there was insufficient memory to perform the procedure.

4 - LC_INVALID - The specified locale is invalid.

5.4.2 procedure freelocale(in lc: locale) returns integer.

The "freelocale" procedure frees the memory occupied by the locale "lc". It returns 0 if the opertion is succesful, and 1 otherwise.

5.4.3 procedure modifylocale(in category: integer, in localename: string, in lc: locale) returns integer.

The "modifylocale" procedure modifies the values of the locale object "lc" relating to the category "category" and with values as specified in "localename". "category" takes values as defined in 4.2 and "localename" is defined as for the "newlocale" procedure. The return values are as for the "newlocale" procedure.

6 Character handling

The character handling procedures behave according to the LC_CTYPE category of the locale parameter for the individual procedures.

6.1 procedure istype(in c: character, in c_type: integer, in lc: locale) returns integer.

The "istype" procedure returns 1 if the character "c" is in the type "c_type", else 0.

"c_type" can have the following values:

alnum, alpha, cntrl, digit, graph, lower, print, punct, space, blank, upper, xdigit

6.2 procedure tolowers(in s: string, in lc: locale) returns string.

The "tolowers" function returns a string with all characters in the string "s" converted to the corresponding lowercase characters with conversion rules given by the locale "lc".

6.3 procedure touppers(in s: string, in lc: locale) returns string.

The "touppers" function returns a string with all characters in the string "s" converted to the corresponding uppercase characters with conversion rules given by the locale "lc".

6.4 procedure stringtrans(in transtype: integer, in maxlen: integer, out s1: string, in s2: string, in rep: repertoire) returns integer.

The "stringtrans" procedure transforms string "s2" into string "s1" given the transformation specifications as noted below.

Values for the "transtype" parameter are

"tolower" - as for the "tolowers" procedure

"toupper" - as for the "touppers" procedure

"transliterate" - transliterate the string "s2" into the string "s1" using for each character the first "transform" specification of ISO/IEC 14652, that are using the repertoire of "rep" and has at most "maxlen" characters as the transliteration. If the "s1" string is to be exceeded, or there is no valid transliteration, the procedure returns -1. Otherwise it returns the resulting number of characters in "s1".

7 String comparison

The string comparison procedures behave according to the LC_COLLATE category of the locale parameter for the individual procedures.

7.1 procedure stringcoll(in s1: string, in s2: string, in precision: integer, in lc: locale) returns integer.

7.2 procedure stringncoll(in s1: string, in s2: string, in precision: integer, in n: integer, in lc: locale) returns integer.

The "stringcoll" procedure compares the two strings "s1" and "s2" with regards to the collating specifications of the locale "lc" and to the precision in "precision".

The "stringncoll" procedure compares at most "n" characters of the two strings "s1" and "s2" with regards to the collating specifications of the locale "lc" and to the precision in "precision".

"precision" may have the following values:

CASE_AND_ACCENT_INSENSITITVE

CASE_INSENSITIVE

IGNORE-SPECIALS

EXACT-MATCHING

Both the "stringcoll" and "stringncoll" procedures returns -1 if "s1" < "s2", 0 if "s1" == "s2" and 1 if "s1" > "s2" .

7.3 procedure stringxfrm(out s1: pointer to octet, in s2: string, in precision: integer, in lc: locale) returns integer.

The "stringxfrm" procedure converts the character string "s2" using the locale "lc" and to the precision in "precision" to an internal representation in "s1" suitable for comparison via a binary comparison function (in C this may be strcmp()).

8 Message formatting

The message formatting procedures behave according to the LC_MESSAGES category of the locale parameter for the individual procedures.

8.1 procedure stringget(in msgtag: string, in textdomain: string, in lc: locale) returns string.

The "stringget" procedure gets the message with the tag "msgtag" in the current LC-MESSAGES part of the "lc" locale with respect to the "textdomain "set of messages. If not found or the locale is invalid and no "msgtag" is found in the default locale, then "msgtag" is returned.

9 Conversion between string and other data types

9.1 procedure string2int(in s: string, in lc: locale) returns integer.

The "string2int" procedure converts a string to an integer, with respect ot the locale "lc".

9.2 procedure int2string(in i: integer, in lc: locale) returns string.

The "int2string" procedure creates a string with the necessary length and returns the string with an integer formatted in characters, according to the locale "lc".

9.3 procedure string2real(in s: string, in lc: locale) returns real.

The "string2real" procedure converts a string to a real value, using information about thousands and decimal separators from the locale "lc".

9.4 procedure real2string(in r: real, in lc: locale) returns string.

The "real2string" procedure formats a real value into a string, with decimal and thousands separators as given in the "lc" locale. Returns a string with the necessary length, or if memory was not available it returns the empty string.

9.5 procedure bytes2string(out s: string, inout p: pointer to sequence of octet, in len: integer, in enc: encoding) returns integer.

The "bytes2string" procedure converts "len" octets from the sequence of octets pointed to by "p" in the encoding "enc" to the string "s", and with the conversion input_state as recorded in "enc". The conversion stops earlier in two cases: if the next character to be stored in the string "s" would exceed the length of "s", or if there is a sequence not corresponding to a recognizable character in the input sequence of octets, possibly after calling an application-defined "get_symbolic_char" procedure.

If the procedure stops without having converted "len" octets, the procedure returns the negative to the number of octets converted. Otherwise it returns the number of internal characters converted (ie. the last index in the string "s" for characters converted).

9.6 procedure string2bytes(out p: pointer to sequence of octet, in s: string, in len: integer, in enc: encoding) returns integer.

The "string2bytes" procedure converts a string "s" into a sequence of corresponding bytes pointed to by "p" in the encoding "enc" , and beginning in the output_state recorded in "enc". The conversion continues up to the lenght of the string "s". The conversion stops earlier in two cases: when a code is reached that does not correspond to a valid representation in the sequence of octets, and either no "invalid_char" value or "put_symbolic_char" procedure is defined, or the application defined "put_symbolic_char" procedure returns with a value 2; or the next octet would exceed the limit of "len" total octets to be stored in the sequence pointed to by "p".

The procedure returns the negative index of the character in question if the conversion stops because it could not convert a character in the string to octets. Otherwise it returns the number of octets in the resulting sequence of octets.

9.7 procedure time2string(out s: string; in format: string; in timeptr: pointer to time, in lc: locale) returns integer.

The "time2string" procedure returns a string "s" formatted according to the format in "format" of the time value in "timeptr", according to the local conventions in the locale "lc". The "format" string is specified in IS 14652 as the "d_t_fmt" specification.

9.8 procedure string2time(out timeptr: pointer to time, in s: string, in lc: locale) returns integer.

The "string2time" procedure returns a binary time in "timeptr", scanned from the string "s" according to the locale conventions in the locale "lc". NOTE: This specification needs more work. The C++ standard is the only standard having provisions for this, but is very weak on the subject.

10 String handling

10.1 procedure stringcopy(out s1: string, in s2: string) returns string.

The "stringcopy" procedure copies the string pointed to by "s2" into the string "s1". If copying takes place between objects that overlap, the behaviour is undefined. The "stringcopy" procedure returns the value of "s1".

10.2 procedure stringncopy(out s1: string, in s2: string, in n: integer) returns string.

The "stringncopy" procedure copies not more than "n" characters (including null characters) from the string "s2" to the string "s1". If string "s2" has a length less than "n", null characters are appended to the copy of "s1" in "s2" until "n" characters in all have been written. The "stringncopy" procedure returns the value of "s1".

10.3 procedure stringcat(in s1: string, in s2: string) returns string.

The "stringcat" procedure appends a copy of the string "s2" to the end of a copy of the string "s1". The "stringcat" procedure returns the value of "s1".

10.4 procedure stringncat(in s1: string, in s2: string, in n: integer) returns string.

The "stringcat" procedure appends not more than "n" characters (including null characters) from the string "s2" to the end of a copy of the string "s1". The "stringncat" procedure returns the value of "s1".

10.5 procedure stringchr(in s: string, in c: character) returns integer.

The "stringchr" procedure locates the first occurrence of "c" in the string "s".

The "stringchr" procedure returns the index in string "s" to the character found, or zero if the character is not found in the string.

10.6 procedure stringcspn(in s1: string, in s2: string) returns integer.

The "stringcspn" procedure returns the length of the maximum initial segment of the string pointed to by "s1" which consists of characters not from the string "s2".

10.7 procedure stringpbrk(in s1: string, in s2: string) returns integer.

The "stringpbrk" procedure locates the first occurence in the string "s1" of any character from the string "s2". The "stringpbrk" procedure returns an index to the first character of the string found, or zero if the string is not found or if "s1" is a string with zero length.

10.8 procedure stringrchr(in s: string, in c: character) returns integer.

The "stringrchr" procedure locates the last occurrence of "c" in the string "s", and returns the index in string "s" to the character found, or zero if the character is not found in the string.

10.9 procedure stringspn(in s1: string, in s2: string) returns integer.

The "stringspn" procedure returns the length of the maximum initial segment of the string pointed to by "s1" which consists of characters from the string "s2".

10.10 procedure stringstr(in s1: string, in s2:string) returns integer.

The "stringstr" procedure locates the first occurrence of the string "s1" in the string "s2", and returns the index in string "s2" to the first character of the string found, or zero if the string is not found or if "s1" is a string with zero lenght.

10.11 procedure stringtok(inout s1: string, in s2: string, inout p: integer) returns integer.

A sequence of calls to the "stringtok" procedure breaks the string "s1" into a sequence of tokens, each of which is delimited by a character from the string "s2". The third argument is an index to string "s1" into which the "stringtok" procedure stores information necessary for it to continue scanning the same wide string.

(informative)

Annex A: Rationale

The specifications in this standard comprise two sets of functionalities:

1. interface to cultural specifications as specificed with ISO/IEC 14652 Cultural Data Specification Format.

2. string handling of the ISO/IEC 10646 - UCS - repertoire

As the specifications in ISO/IEC 14652 is declared to be upwards compatible with similar specifications in ISO/IEC 9945-2 POSIX Shell and Utilites, which in turn built on functionality in the ISO/IEC 9899 C standard, the specifications in this standard are modelled after these standards.

Changes that has been necessary to make compared to the IS 9899 and IS 9945 standards have been:

1. To be able to operate in an environment where several light-weight processes (also known as "threads") can be run, it has been necessary to avoid using a "global locale", and all locale dependent interfaces have the locale as a parameter.

2. To be able to have the locale as a parameter, a "locale" data type has been defined. This contains the C "lconv" struct as one of its data.

3. To be able to have bindings from the Language-Independent specification to other language families than the C family, strings have been allowed to have null-characters in them, and thus a length attribute has been introduced as an integral part of the string data type. The introduction of a separate string data type, which is not terminated by a null character, has introduced a number of changes to the C-like APIs. In particular it has not been possible to define a return value as just a pointer into a string, and in the cases where a pointer was returned in C, the LI APIs return an index into a string.

4. The "tolowers" and "touppers" functions have been defined on strings instead of characters to allow conversions like German "eszet" to be converted to two characters, and to facilitate conversions of full strings.

5. The conversion functions have been modelled so that there is a conversion to and a conversion from each of the other major data types.

6. In the conversion between the internal string data type and external character representation, the conversion procedures have been greatly enhanced with a number of extra capabilities.

7. String collation has been done according to IS 14651, which added the precision parameter and also the locale parameter.

(normative)

Annex B: C binding

This annex specifies the binding of this International Standard to the C Programming Language.

The C binding to procedures defined in language-independent (LI) notation is done by a reference to the section number where the procedure is defined, and then the C function name is bound to the LI procedure, the C parameters are bound to the LI parameters in the same sequence, and the C return value is bound to the LI return value.

The string type is bound to a pointer to a wchar_t array with the first element giving the length (in characters) of the string.

To use the C functions a header <i18n.h> is defined with the appropiate function and macro declarations.

Data types creation, deletion and attribute functions:

5.1.1 string newstring(const int length)

5.1.2 int freestring(const string s)

5.1.3 int stringlen(const string s)

5.2.1 int newencoding(const string encodingname, struct encoding *enc)

5.2.2 int freeencoding(const struct encoding *enc)

5.2.3 int setencint(const struct encoding *enc, const string param, const int val)

5.2.4 int setencbytes(const struct encoding *enc, const string param, const char *val, const int len)

5.2.5 int setencproc(const struct encoding *enc, const string param, const int proc())

5.3.1 int newrepertoire(const string repertoirename, struct repertoire *rep)

5.3.2 int freerepertoire(const struct repertoire *rep)

5.3.3 int enc2repertoire(const struct encoding *enc, struct repertoire *rep)

5.4.1 int newlocale(const int category, const string localename, struct locale *lc)

5.4.2 int freelocale(const struct locale *lc)

5.4.3 int modifylocale(const int category, const string localename, const struct locale *lc)

Cultural handling:

6.1 int istype(int c, int c_type, struct locale *lc);

6.2 int tolowers(string s, struct locale *lc);

6.3 int touppers(string s, struct locale *lc);

6.4 int stringtrans(const int transtype, const int maxlen, string s1, const string s2, const struct repertoire *rep)

7.1 int stringcoll(const string s1, const string s2, int p, struct locale *lc);

7.2 int stringncoll(const char *s1, const string s2, int p, int n, struct locale *lc);

7.3 size_t stringxfrm(char *s1, const string s2, int n, struct locale *lc);

8.1 stringget(string msgtag, string textdomain, struct locale *lc)

Conversion between the string type and other types:

9.1 string int2string(long i, const struct locale *lc)

9.2 long string2int(const string s, const struct locale *lc)

9.3 string real2string(const double r, const struct locale *lc)

9.4 double string2real(const string s; const struct locale *lc)

9.5 int bytes2string(string s, char *p, const in len, const struct encoding *enc)

9.6 int string2bytes(char *p, constr string s, const int len, const struct encoding *enc)

9.7 int time2string(string s, size_t maxsize, const string format, const struct tm *timeptr, struct locale *lc);

9.8 int string2time(string s, size_t maxsize, const string format, const struct tm *timeptr, struct locale *lc);

String handling:

10.1 string stringcpy(string s1, const string s2);

10.2 string stringncpy(string s1, const string s2, size_t n);

10.3 string stringcat(string s1, const string s2);

10.4 string stringncat(string s1, const string s2, size_t n);

10.5 int stringchr(const string s, int c);

10.6 int stringcspn(const string s1, const string s2);

10.7 string stringpbrk(const string s1, const string s2);

10.8 int stringrchr(const string s, int c);

10.9 int sringspn(const string s1, const string s2);

10.10 int stringstr(const string s1, const string s2);

10.11.int stringtok(string s1, const string s2);