ISO/IEC JTC1/SC22/WG20 N_____

Date: 2002-04-16

ISO/IEC 15435.7

ISO/IEC JTC1/SC22/WG20

Secretariat:ANSI

Information technology - Internationalization APIs

Technologies de l'information - IPA concernant l'internationalisation

Contents

Foreword

ISO (the International Organization for Standardardization) 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 TR 14652.

Annex B is normative.

Annex A and C are informative.

This is WD7 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.

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 TR 14652, and string handling. An API in the form of a utility to handle descriptions of TR 14652 is specified.

2 Conformance

A conforming binding is a language binding that binds to all the APIs in this standard, except application defined and utility APIs, and with semantics as specified in this standard. For APIs with more than 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 9899:1999, *Information Technology - Programming language C*

ISO/IEC 9945-2:1993*, Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities.*

ISO/IEC 10646-1:2000, *Information Technology - Universal Multiple-Octet Coded Character Set (UCS)*

ISO/IEC FDIS 14651, *Information Technology - International string ordering - Method for comparing character strings and description of the common template tailorable ordering*

ISO/IEC 15897:1999, *Information Technology - Procedures for 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.

NORE: The APIs are described in the form of a function or utility description of a API 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.

4.4
internal character representation
representation of characters for use internally in a program.

4.5
external character representation
representation of characters for use externally to a program, for example in files or as input or output to communication devices.

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 length. The string type can hold all characters of ISO/IEC 10646 including null characters -The string encoding shall be independent of the locale. All length information is given in terms of characters (not storage units). An empty string is indicated by a string with the lenght 0. A void string is indicated by the implementation-defined value NIL. The string data type is defined by:

```
#define string *wchar_t
```

5.2 Encoding data type

The "encoding" data type holds data necessary to convert to and from an external encoding and 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. The enocding data type is defined by:

```
struct encoding {

        string encodingname;

        // other things ???

}
```

NOTE The encoding definition is closely related to the "charmap" specification of POSIX and TR 14652, the "charset" definition in the Internet MIME specification, and newer developments for the C and C++ programming languages.

5.2.1 int newencoding(const string encodingname,encoding enc)

The "newencoding" API 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" API sets default values, including the "line_terminator" to being the characters "CR""LF", the "invalid_char" to being the character "SUB", the "symbolic_char_introducer" to being "NUL" (not valid), the "sub_encoding_change" API, the "get_symbolic_char_name" API and the "put_symbolic_char_name" API to be the null API, 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, ISO/IEC 15897.

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

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

0 -LC_SUCCESS - The API call was successful

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

2 - LC_NOMEMORY - there was insufficient memory to perform the API

3 -LC_INVALID - The specified encoding is invalid

5.2.2 int freeencoding(encoding enc)

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

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

iThe "setencint" API 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 values for "param" are:

(to be described)

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

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

The "setencbytes" API 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 values for "param" are:

"line_terminator"
"invalid_char"
"symbolic_char_introducer"

5.2.5 int setencproc(encoding enc,const string param,int val())

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

"sub_encoding_change" - procedure subec()
"get_symbolic_char_name" - procedure gscn(c, p , len)
"put_symbolic_char_name" - procedure pscn(c, p, len)

5.2.5.1 subec()- a API with no parameters (NOTE, is this true??)

?? Comes from Ian MacLeod

5.2.5.2 int gscn(c, p, len)

The application defined "get_symbolic_char_name" API is called by the "bytes2string" API 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 APIs definitions, with or without a terminator sequence, within "len" octets after the "p" pointer. If successful the API 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 API 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.

5.2.5.3 int pscn(c, p, len)

The application defined "put_symbolic_char_name" API is called by the "string2bytes" API 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 APIs definitions, with the "symbolic_char_introducer" value and with or without a terminator sequence, within "len" octets after the "p" pointer. If successful, the API returns "p", a pointer, to the first octet after the symbolic character written, including the possible terminator sequence. The application defined API returns:

0 if successful

1 if the API was not able to write the symbolic character within "len" octets. The pointer "p" is not changed.

2 if the API had no means of writing the character "c", The pointer "p" is not changed.

5.3 Repertoire data type

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

5.3.1 int newrepertoire(const string repertoirename,repertoire rep)

The "newrepertoire" API creates a repertoire object with the necessary space to hold all information necessary. The "repertoirename" is an implementation defined string with the following characteristics: An initial string of "std/" refers to repertoiremaps registered in the international cultural register, ISO/IEC 15897. If the specified repertoire is valid and supported, the "newrepertoire" API 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" API when the object is no longer needed. If the API fails for any reason, the contents of "rep" is undefined.

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

0	- The API call was successful
1	- The repertoire is not supported by the current system.
2	- There was insufficient memory to perform the API
3	- The specified repertoire is invalid

5.3.2 int freerepertoire(repertoire rep)

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

5.3.3 int enc2repertoire(encoding enc,repertoire rep)

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

5.4 Locale data type

The "locale" data type is a pointer to a struct 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:

1	NULL
2	LC_ALL
3	LC_IDENTIFICATION
4	LC_COLLATE
5	LC_CTYPE
6	LC_MONETARY
7	LC_NUMERIC
8	LC_TIME
9	LC_MESSAGES
10	LC_XLITERATE
11	LC_NAME
12	LC_ADDRESS
13	LC_TELEPHONE

The category LC_ALL denotes all of the other non-void categories.

The category NULL denotes a void category.

The "locale" data type includes the following variables (which are further described in ISO/IEC TR 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 (1,13) of string
mon: array (1,13) 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 int newlocale(int category_mask, const string localename,locale lc)

The "newlocale" API creates a locale struct with all the necessary information to perform the language-sensitive operations of internationalization APIs accepting an argument of the type "locale". If the API is successful, all categories of the locale object are created and initialized. Any categories in the locale identified by "localename" are initalized to the i18n locale.

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

- An initial string of "std/" refers to the locales registered in the international cultural register, ISO/IEC 15897.

If the specified locale is valid and supported, the "newlocale" API 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" API when the object is no longer needed. If the API fails for any reason, the contents of "lc" is undefined.

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

0 - LC_SUCCESS - The API 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 i18n 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 API.

4 - LC_INVALID - The specified locale is invalid.

5.4.2 int freelocale(locale lc)

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

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

The "modifylocale" API modifies the values of the locale object "lc" parameter relating to the category "category" and with values as specified in "localename". "category" takes values as defined in 5.4 and "localename" is defined as for the "newlocale" API. The return value is as for the "newlocale" API.

5.4.4 int intlocaleinfo(const int category,const string keywordname,locale lc)

The "intlocaleinfo" API gets the integer value of the keyword "keywordname" of the locale object "lc" relating to the category "category". "category" takes values as defined in 5.4.The return value is the integer value of the keyword.

5.4.5 string stringlocaleinfo(const int category,const string keywordname,locale lc)

The "stringlocaleinfo" API gets the string value of the keyword "keywordname" of the locale object "lc" relating to the category "category". "category" takes values as defined in 5.4.The return value is the string value of the keyword.

6 Character handling

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

6.1 int istype(wchar_t c,const string c_type,const locale lc)

The "istype" API 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 int tolowers(string s1,const string s2,const locale lc)

The "tolowers" API returns in string "s1" all characters in the string "s2" converted to the corresponding lowercase characters with conversion rules given by the locale "lc". The API returns the number of resulting characters in "s1".

6.3 int touppers(string s1,const string s2,const locale lc)

The "touppers" API returns in string "s1" all characters in the string "s2" converted to the corresponding uppercase characters with conversion rules given by the locale "lc". The API returns the number of resulting charactes in "s1".

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

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

Values for the "transtype" parameter are

1 - as for the "tolowers" API

2 - as for the "touppers" API

3 - transliterate the string "s2" into the string "s1" (for example using for each character the first "transform" specification of ISO/IEC TR 14652) that is 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 API returns -1. Otherwise it returns the resulting number of characters of "s1".

7 String comparison

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

7.1 int strcoll_l(const string s1,const string s2,locale lc)

7.2 int strncoll_l(const string s1,const string s2, n,locale lc)

The "strcoll_l" API compares the two strings "s1" and "s2" with regards to the collating specifications of the locale "lc".

The "strncoll_l" API compares at most "n" characters of the two strings "s1" and "s2" with regards to the collating specifications of the locale "lc".

Both the "strcoll_l" and "strncoll_l" APIs returns -1 if "s1" < "s2", 0 if "s1" == "s2" and 1 if "s1" > "s2" .

7.3 int strxfrm_l(const string s1,const string s2,locale lc)

The "stringxfrm" API converts the character string "s2" using the locale "lc" and to the precision in "precision"as defined in 7.2, to an internal representation in "s1" suitable for comparison via a binary comparison API (in C this may be strcmp()).

7.4 int stringcoll(const string s1,const string s2,int precision,locale lc)

7.5 int stringncoll(const string s1,const string s2,int precision,int n,locale lc)

The "stringcoll" API 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" API 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".

The "precision" indicates to what level of preciseness the string comparison is done. "precision" may have the following values:

0 - all levels

1 - only to level 1 - CASE_AND_ACCENT_INSENSITITVE

2 - only to level 2 - CASE_INSENSITIVE

3 - only to level 3 - IGNORE_SPECIALS

4 - only to level 4 - EXACT_MATCHING

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

7.6 int stringxfrm(const string s1,const string s2,int precision,locale lc)

The "stringxfrm" API converts the character string "s2" using the locale "lc" and to the precision in "precision" as defined in 7.2, to an internal representation in "s1" suitable for comparison via a binary comparison API (in C this may be strcmp()).

8 Message formatting

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

8.1 string stringget(const string msgtag,const string textdomain,locale lc)

The "stringget" API 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 int string2int(string s,locale lc)

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

9.2 string int2string(int i,locale lc)

The "int2string" API creates a string with the necessary length and returns the string with an integer formatted in characters, according to the locale "lc". If there is not enough memory to create a new string, the API returns the void string.

9.3 double string2real(string s,locale lc)

The "string2real" API converts a string to a real value, using information about thousands and decimal separators from the locale "lc". If there is not enough memory to create a new string, the API returns the void string.

9.4 string real2string(double r,locale lc)

The "real2string" API 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 is not available it returns the empty string.

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

The "bytes2string" API converts "len" octets from the multibyte value "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" API.

If the API stops without having converted "len" octets, the API 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 int string2bytes(char* p,string s,int len,encoding enc)

The "string2bytes" API converts a string "s" into a sequence of corresponding octets of "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" API is defined, or the application defined "put_symbolic_char" API returns with a value 2; or the next octet would exceed the limit of "len" total octets to be stored in the multibyte "p" variable.

The API 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 int time2string(string s,const string format,const struct tm *timeptr,locale lc)

The "time2string" API 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 9945 (with extensions as described in TR 14652) as the "d_t_fmt" specification.

9.8 int string2time(const struct tm *time,string s,locale lc)

The "string2time" API returns a binary time in "time", 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.

9.9 int money2string(string s,const string format,const double amount,const struct tm *timeptr ,locale lc)

API "money2string" returns in parameter "s" a string formatted according to the format in "format" of the money value "amount". The formatting is done with respect to the locale "lc" at the time given in "time". The return value is the number of characters formatted, or -1 if an error occurred.

The parameter "format" is a string that consist of characters that shall be transfered to the output string "s" literally, and formatting specifications that specifies how the money value "amount" is to be formatted.

A formatting specification consist of the following sequence:

- a "%" character
- optional flags
- optional field width
- optional left precision
- optional right position
- the formatting character to specify which formatting to perform.

Flags are given with special characters, width and precision information is given with decimal digits, and formating characters are given with latin letters or the character "%".

The flags are:

"=f" an "=" followed by a single character which is used as the numeric fill character. No restriction is made on the representation of this single character. The default numeric fill character is the SPACE character. This flag does not affect the field width filling which always uses the SPACE character. This flag is ignored unless a left precision (se below) is specified.

"^" Do not use the grouping characters when formatting the amount. The default is to insert the grouping characters if defined in the locale "lc".

"+" or "(" Specify the style of representing positive and negative amounts. Only one of "+" or "(" may be specified. If "+" is specified, the equivalent of "+" and "-" are used from the locale "lc". If "(" is specified, negative amounts are enclosed within parenthesis. If neither flag is specified, the "+" style is used.

"!" Suppersses the currency symbol from the output conversion.

"-" Specify the alignment. If this flag is present all fields are left-justified (padded to the right) rather than right-justified.

Field Width

w A string of decimal digits specifying the minimum field width in characters in which the result of the conversion is right-justified (or left-justified if the "-" flag is specified) The default is 0.

Left Precision

"#n" A "#" followed by a string of decimal digits specifying a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from several calls to API "money2string" aligned in the same coloumns. It can also be used to fill unused positions with a special character specified with the "=f" flag, as in $***123.45. If more than "n" positions are required, this formatting specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character, see the "=f" flag above.

If grouping has not been suppressed with the "^" flag, and it is defined for the locale "lc", grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters, even if the fill character is a digit.

To ensure alignment, any characters appearing before or after the number in the formatted output such as currency or sign, symbols are padded as neccessary with SPACE characters to make their positive or negative formats an equal length.

Right precision

".p" A "." followed by a string of decimal digits specifying the number after the radix character. If the value of the right precision is 0, no radix character appears. If a right precision is not included, the value specified in the "lc" locale is used. It is recommended to normally use the value from the locale. The amount being formatted is rounded to the specified number of digits prior to formatting.

Formatting characters:

The formatting characters and their meanings are:

"d" The following characters and up to any corresponding "%d" or the end of the formatting string are only interpreted if there is a second currency in the "lc" locale for the time "t". The amount "a" is converted according to the "conversion_rate" of the locale "lc" and formatted according to any following formatting characters. No argument is converted. There shall be no flags, nor width or precision parameters, just "%%" is allowed.

"i" The type money argument is formatted acording to the "lc" locales's international currency format.

"n" The type money argument is formatted acording to the "lc" locales's national currency format.

"%" Convert to a "%"; no argument is converted. There shall be no flags, nor width or precision parameters, just "%%" is allowed.

9.10 int name2string(string s,const string format,const string name,locale lc)

The API name2string formats a set of personal name information as given in "name" to a string "s" according to the format in"format" and to the locale given in "lc". The format specification is unspecified (but a description may be found in TR 14652 for the keyword "name_fmt") if this is the empty string, the format specified in the "name_fmt" keyword of the locale in "lc" is used. The return value is the number of characters in the resulting string "s" or -1 if the supplied string "s" had insufficient length to hold the result.

The namerecord shall contain the following strings, which may each be empty:

family - family names, corresponding to the %f and %F escape sequence
given - first given name
giveninit - initial of given name
middle - middle names
middleinit - middle initials
shortname - a shorter name, eg. "Bill"
profession - the profession title
salutation - common salutation, like "Mr."
intsalut - a string with a digit in the range 1 to 5 for salutation

Similar strings with a "r" prepended to the name shall be present to hold Romanized information on the above items.

9.11 int address2string(string s,const string format,const string address,locale lc)

API address2string formats a set of address information as given in "address" to a string "s" according to the format in"format" and to the locale given in "lc". The format specification is unspecified, (but a description may be found in TR 14652 for the keyword "postal_fmt"); if this is the empty string, the format specified in the "postal_fmt" keyword of the locale in "lc" is used. The return value is the number of characters in the resulting string "s" - or -1 if an error occurred.

The addressrecord shall contain the following strings (with corresponding escape sequences given in parentheses), which may each be empty:

name - person's name (%n)
co - C/o address (%a)
firm - firm name (%f)
department - department name (%d)
building - building name (%b)
streetblock - street or block name (%s)
house - house number or designation (%h)
room - room number or designation (%r)
floor - floor number (%e)
village - village name (%v)
town - town or city name (%T)
countrycode - country designation or code (%C)

zip - zip or postal code (%z)
country- country name (%c)

Similar strings with a "r" prepended to the name shall be present to hold Romanized information on the above items.

9.12 int teldom2string(string s,const string format,const string telephone,locale lc)

API teldom2string formats for domestic use a telephone number as given in "telephone" to a string "s" according to the format in"format" and to the locale given in "lc". The format specification is unspecified (but a description may be found in TR 14652 for the keyword "tel_dom_fmt"); if this is the empty string, the format specified in the "tel_dom_fmt" keyword of the locale in "lc" is used. The return value is the number of characters in the resulting string "s" - or -1 if an error occurred..

9.13 int telint2string(string s,const string format,const string telephone,locale lc)

The API telint2string formats for international use a telephone number as given in "telephone" to a string "s" according to the format in"format" and to the locale given in "lc". The format specification is unspecified (but a description may be found in TR 14652 for the keyword "tel_int_fmt"); if this is the empty string, the format specified in the "tel_int_fmt" keyword of the locale in "lc" is used. The return value is the number of characters in the resulting string "s" - or -1 if an error occurred.

10 Utilities

Utilities are APIs that provide an interface at runtime, as a program.

10.1 localedef

localedef [-c] [-f charmap [-F char-repertoire]] [-i locale-source [-I locale-repertoire]] localename

The "localedef" utility shall convert source definitions for locale or FDCC-set categories into a format usable by the APIs and utilities whose operational behaviour is determined by the locale.

The utility shall read source definitions for one or more categories from the file named in the "-i" option (if specified) or from the standard input.

The "localename" identifies the target locale.

The following options shall be supported by the implementation:

-c Create permanent output even if warning messages have been issued.

-f charmap Specify the pathname of a file containing a mapping of character symbols and collating element symbols to actual character encoding. The format of the charmap is described in IS 9945 (with possible extentions described in TR 14652). This option shall be specified if symbolic names (other than collating symbols defined in the input locale or FDCC-set) are used. If the "-f" option is not present, an implementation-defined character mapping is used.

-F char-repertoire The pathname of a file containing a repertoiremap describing mapping between character symbols used in the charmap and IS 10646 characters.

-i locale-source The pathname of a file containing the source defitions of the categories. If this option is not present, shource definitions shall be read from standard input. The format of the FDCC-set and locale is described in IS 9945 (with possible extensions described in TR 14652).

-I locale-repertoire The pathname of a file containing a repertoiremap describing mapping between character symbols used in the locale and IS 10646 characters.

The -F and -I options need only be specified if the use of character symbols differ between the locale-source and charmap.

The following operand shall be supported by the implementation

localename Identifies the output locale. If the name contains one or more <solidus> characters, "localename" shall be interpreted as a pathname where the created locale definiton(s) shall be stored. If "localename" does not contain any <solidus> characters, the interpretation of the name is implementation defined, and the locale shall be public. This capability may be restricted to users with appropriate privileges.

The utility shall report all categories successfully processed, in an unspecified format

The format of the created output file is unspecified.

The utility shall exit with one of the follwing values:

0 No errors occurred and the output files were sucessfully created

1 Warnings occurred and the output fileswere successfully created

2 The locale specifications exceeded implementation limits, or the charmap used was not supported by the implementation, and no output files were created.

>3 Warnings or errors occurred and no output files were created.

Consequence of errors

If an error is detected, no permanent output shall be created.

If warnings occur, permanent output shall be created if the "-c" option was specified. The following conditions shall cause warning messages to be issued:

- If a symbolic name used in the LC_CTYPE or LC_COLLATE categories cannot be matched to a corresponding symbolic name in the "charmap" (for other catergories, this shall be an error condition). The match is true if the symbolic name is found both in the source locale and in the charmap; or if a locale-repertoire and char-repertoire file is specified, the match is true if there exist a symbolic name in each of the repertoiremaps that match to the same character in IS 10646.

- If the number of operands to the "order_start" keyword exceeds the COLL_WEIGHTS_MAX limit

- If optional keywords not supported by the implementation are present in the source.

Other implementation-defined conditions may also cause warnings.

Annex A:

(informative)

Rationale

11A.1 Introduction

The specifications in this standard comprise two sets of functionalities:

1. interface to cultural specifications as specificed with ISO/IEC 9945 and with the extensions described in ISO/IEC TR 14652 Specification method for cultural conventions.

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

12A.2 Relations between some character set terms

The encoding of a character repertoire can consist of at least three parts:

1. The (simple, composite or full) coded character sets, for example ISO/IEC 8859-1 combined with the control character set of ISO/IEC 6429.

2. The rules for combining or coding one or more simple coded character sets, for example ISO/IEC 2022.

3. A symbolic character notation, like SGML entities of the type &aacute;

On top of the character repertoire more complex "text elements" may be composed consisting of one or more (abstract) characters, for example text elements of Indic script characters.

On top of the encoding, general transformation schemes that are applicable to any binary representation may be applied. These generally applicable schemes include:

* compression schemes, (examples: zip, gzip)

* encryption schemes, (examples PGP)

* safer passage schemes, such as avoiding the 8th bit or byte values 0-31, (Examples: base64, uuencode)

Editors note: a figure is probably useful here.

13A.3 Enhancements from IS 9899 and IS 9945 series standards

As the descriptions in ISO/IEC TR 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 those 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. The "tolowers" and "touppers" APIs 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.

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

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

6. String collation has been designed to use data from to IS 14651, including a "precision" parameter.

7. money2string rationale:

The money2string function needs to have a number of capabilities:

1. functionality of X/Open and C++: strfmon and money_put
2. exact mode - integer 64 bits
3. handle euro in transparant way
4. thread-safe - needs locale parameter
5. Interfaces to data as described in TR 14652.

Point 3 needs further elaboration. In countries with the Euro it is required by law in a period of 2 1/2 years to display both the national currency and the Euro. The program should be the same whether it is intended say for USA or Denmark that do not have the Euro, or France and Germany which has the Euro. Also there should not be a need to change locales when a country shifts from national currency to the dual currency, nor when the country changes to just Euro.

Example of use (in C):

```
  modifylocale(LC_ALL,"de_DE",lc); money2string(s,"%i%d %i",123456,19990601,lc);
  may format "s" to contain "DEM 1.234,56 EUR 433,84"

  modifylocale(LC_ALL,"en_US",lc); money2string(s,"%i%d %i",123456,19990601,lc);
  may format "s" to contain "USD 1,234.56"
```

A.4 Impementation of this standard in glibc

Editors note:This clause may be removed in the final standard.

If the API has been implemented in glibc, it is marked with a "g" in front of the section number.

Data types creation, deletion and attribute APIs:

```
  5.2.1 int newencoding(const string encodingname, encoding enc)
  5.2.2 int freeencoding(encoding enc)
  5.2.3 int setencint(encoding enc, const string param, int val)
  5.2.4 int setencbytes(encoding enc, const string param, const char *val, int len)
  5.2.5 int setencproc(encoding enc, const string param, int proc())
  5.3.1 int newrepertoire(const string repertoirename, repertoire rep)
  5.3.2 int freerepertoire(const repertoire rep)
  5.3.3 int enc2repertoire(const encoding enc, repertoire rep)
g 5.4.1 locale newlocale(int category, const string localename, locale lc)
g 5.4.2 void freelocale(locale lc)
  5.4.3 int modifylocale(const int category, const string localename, locale lc)
  5.4.4 int intlocaleinfo(const int category, const string keywordname, locale lc)
g 5.4.5 string nl_langinfo_l( const string keywordname, locale lc)
```

also in glibc:

```
g locale duplocale(locale lc)
g string setlocale(int category, string localename)
```

Cultural handling:
g 6.1 int iswctype_l(wint_t wc, int c_type,locale lc)
  6.2 int tolowers(string s, locale lc)
  6.3 int touppers(string s, locale lc)
  6.4 int stringtrans(const int transtype, const int maxlen, string s1, const string s2, repertoire rep)
g 7.1 int wcscoll_l(const string s1, const string s2, locale lc)
g 7.2 int wcsncoll_l(const string s1, const string s2, int n, locale lc)
g 7.3 size_t wcsxfrm_l(char *s1, const string s2, int n, locale lc)
  8.1 string stringget(string msgtag, string textdomain, locale lc)

Conversion between the string type and other types:
  9.1 string int2string(long i, locale lc)
  9.2 long string2int(const string s, locale lc)
  9.3 string real2string(const double r, locale lc)
  9.4 double string2real(const string s; locale lc)
  9.5 int bytes2string(string s, char *p, int len, encoding enc)
  9.6 int string2bytes(char *p, constr string s, int len, encoding enc)
  9.7 int time2string(string s, size_t maxsize, const string format, const struct tm *timeptr, locale lc)
  9.8 int string2time(string s, size_t maxsize, const string format, struct tm *timeptr, locale lc)
  9.9 int money2string(string s, const string format, const double money, const struct tm *timeptr, locale lc)
  9.10 int name2string(string s, const string format, const struct namerecord *name, locale lc)
  9.11 int address2string(string s, const string format, const struct addressrecord *address, locale lc)
  9.12 int teldom2string(string s, const string format, const string telephone, locale lc)
  9.13 int telint2string(string s, const string format, const string telephone, locale lc)

(informative)

Annex B: Bindings guidelines

This annex gives guidelines for binding to other programming languages.

Originaly this standard was drafted as a language independent specification (LIS), with a binding to Programming Language C in a normative annex. The standard was then redrafted as a specification in C.

Another programming language (PL) binding to the definition in C notation may be done by a reference to the section number where the procedure is defined, and then the PL API name is bound to the C API, the PL parameters are bound to the C parameters in the same sequence, and the PL return value is bound to the C result value. This clause may then be used as part of the specification of the binding technique

Exampe of a Pascal binding, given the appropiate definitions of data structures:

```
5.4.1 integer procedure newlc(integer cat;string lcname;locale lc)
```

The string type should be bound to the appropiate internal character representation in the PL.

.

Annex C

(informative)

Relation between 14652 and 15435

The relations between the 3 major sections of TR 14652, FDCC-sets, charmaps and repertoiremaps, and 15435 APIs are given in the following.

Table: Relation between 14652 categories and keywords, and 15435 APIs

14652 category	14652 keyword	15435 API
LC_ALL	all keywords	intlocaleinfo stringlocaleinfo
LC_CTYPE	upper lower alpha	istype
	digit outdigit space	istype
	cntrl punct graph	istype
	print xdigit blank	istype
	toupper	touppers stringtrans
	tolower	tolowers stringtrans
	class	istype
	map	istype
LC_COLLATE	all	stringcoll stringncoll
LC_MONETARY	all	money2sting
LC_NUMERIC	all	real2string string2real
LC_TIME	all	time2string string2time
LC_MESSAGES	all	stringget
LC_XLITERATE	all	stringtrans
	translit_start translit_end	stringtrans
	include default_missing	stringtrans
LC_TELEPHONE	tel_dom_fmt	teldom2string
	tel_int_fmt	telint2string
LC_IDENTIFICATION	all	stringlocaleinfo

APIs addressing charmaps:
APIs addressing repertoiremaps:

Bibliography

ISO/IEC TR 14652 (draft) *Information technology - Specification method for cultural conventions*