Technical Standard

Extended API Set Part 4

The Open Group

Technical Standard

Extended API Set Part 4

ISBN: 1-931624-69-0
Document Number: C065

Published in the U.K. by The Open Group, October 2006.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Thames Tower
37-45 Station Road
Reading
Berkshire, RG1 1LX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

Preface

The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including UNIX certification.

Further information on The Open Group is available at *www.opengroup.org*.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

More information is available at *www.opengroup.org/certification*.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at *www.opengroup.org/bookstore*.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards-compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at *www.opengroup.org/corrigenda*.

This Document

This document has been prepared by The Open Group Base Working Group. The Open Group Base Working Group is considering submitting a number of API sets to the Austin Group as input to the revision of the Base Specifications, Issue 6.

This is the fourth document in that set.

Trademarks

Acknowledgements

The contributions of the following to the development of this document are gratefully acknowledged:

- The Open Group Base Working Group

Chapter 1

Introduction

1.1 Scope

The purpose of this document is to define a set of new API extensions to further increase application capture and hence portability for systems built upon the Single UNIX Specification, Version 3.

This proposal adds a set of interfaces that allow applications to use multiple locales concurrently and allow multi-threaded applications to use a different base locale in each thread.

1.2 Relationship to Other Formal Standards

This Technical Standard is being forwarded to the Austin Group for consideration as input to the revision of the Base Specifications, Issue 6.

It is recommended that these functions be integrated as follows:

- When an implementation claims support of this option, all functions except *uselocale*( ) shall be provided.
- If an implementation claims to support both the new option and the Threads option, it must also provide *uselocale*( ).

Chapter 2

Changes to the Base Definitions Volume

It is proposed that these additions comprise a new option called the Multiple Concurrent Locales option.

2.1 Section 1.5.1, Codes

Add a new margin code as follows:

MCL Multiple Concurrent Locales

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the MCL margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MCL margin legend.

Notes:

1. This section is repeated in XBD, XSH, and XCU and therefore will appear in XBD (Section 1.5.1), XSH (Section 1.8.1), and XCU (Section 1.8.1).
2. The use of MCL as a margin code is a placeholder and may change in the final publication.

2.2 Chapter 13, Headers

The following header file reference pages will need the following additions:

<ctype.h>

The following will be added:

MCL The **<ctype.h>** header shall provide a definition for a type **locale_t** as defined in **<locale.h>** representing a locale object.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int isalnum_l(int, locale_t);
int isalpha_l(int, locale_t);
int isblank_l(int, locale_t);
int iscntrl_l(int, locale_t);
int isdigit_l(int, locale_t);
int isgraph_l(int, locale_t);
int islower_l(int, locale_t);
int isprint_l(int, locale_t);
int ispunct_l(int, locale_t);
int isspace_l(int, locale_t);
int isupper_l(int, locale_t);
int isxdigit_l(int, locale_t);
int tolower_l(int, locale_t);
int toupper_l(int, locale_t);
```

<locale.h>

The following will be added:

MCL The **<locale.h>** header shall contain at least the following macros representing bitmasks for use with the *newlocale*( ) function for each supported locale category:

```
LC_COLLATE_MASK
LC_CTYPE_MASK
LC_MESSAGES_MASK
LC_MONETARY_MASK
LC_NUMERIC_MASK
LC_TIME_MASK
```

Implementations may add additional masks using the form LC_*_MASK.

In addition, a macro to set the bits for all categories set shall be defined:

```
LC_ALL_MASK
```

The **<locale.h>** shall define LC_GLOBAL_LOCALE, a special locale object descriptor used by the *uselocale*( ) function.

The <locale.h> header shall provide a definition for a type **locale_t** representing a locale object.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
locale_t newlocale (int, const char *, locale_t);
locale_t duplocale (locale_t);
void freelocale (locale_t);
locale_t uselocale (locale_t);
```

<monetary.h>

The following will be added:

MCL The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
ssize_t strfmon_l(char *restrict, size_t, locale_t,
    const char *restrict, ...);
```

<string.h>

The following will be added:

MCL The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int strcoll_l(const char *, const char *, locale_t);
size_t strxfrm_l(char *restrict, const char *restrict,
    size_t, locale_t);
```

<strings.h>

The following will be added:

MCL The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int strcasecmp_l(const char *, const char *, locale_t);
int strncasecmp_l(const char *, const char *, size_t, locale_t);
```

<unistd.h>

The following will be added:

MCL _POSIX_MULTIPLE_LOCALES
The implementation supports the Multiple Concurrent Locales option. If this symbol is defined in **<unistd.h>**, it shall be defined to be –1, 0, or 200ymmL. The value of this symbol reported by *sysconf*( ) shall be either –1 or 200ymmL.

Note: 200ymmL is to be replaced by the year and month of approval of the standard.

The following will be added to the list of symbolic constants that shall be defined for *sysconf*( ):

_SC_MULTIPLE_LOCALES

<wchar.h>

The following will be added:

MCL The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int wcscasecmp_l(const wchar_t *, const wchar_t *, locale_t);
int wcsncasecmp_l(const wchar_t *, const wchar_t *, size_t, locale_t);
int wcscoll_l(const wchar_t *, const wchar_t *, locale_t);
size_t wcsxfrm_l(wchar_t *restrict, const wchar_t *restrict,
    size_t, locale_t);
```

<wctype.h>

The following will be added:

MCL The **<ctype.h>** header shall provide a definition for a type **locale_t** as defined in **<locale.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int iswalnum_l(wint_t, locale_t);
int iswalpha_l(wint_t, locale_t);
int iswblank_l(wint_t, locale_t);
int iswcntrl_l(wint_t, locale_t);
int iswdigit_l(wint_t, locale_t);
int iswgraph_l(wint_t, locale_t);
int iswlower_l(wint_t, locale_t);
int iswprint_l(wint_t, locale_t);
int iswpunct_l(wint_t, locale_t);
int iswspace_l(wint_t, locale_t);
int iswupper_l(wint_t, locale_t);
int iswxdigit_l(wint_t, locale_t);
int iswctype_l(wint_t, wctype_t, locale_t);
wint_t towctrans_l(wint_t, wctrans_t, locale_t);
wint_t towlower(wint_t, locale_t);
wint_t towupper(wint_t, locale_t);
wctrans_t wctrans_l(const char *, locale_t);
wctype_t wctype_l(const char *, locale_t);
```

Chapter 3

Changes to the System Interfaces Volume

3.1 Changes to Exising Reference Pages

strerror( )

The following changes will be made to the *strerror*( ) reference page (Page 1441).

Change the NAME section to:

strerror, strerror_l, strerror_r — get error message string

The following will be added between the current entries for *strerror*( ) and *strerror_r*( ) in the SYNOPSIS section:

MCL `char *strerror_l(int` *errnum*, locale_t *locale*);

The following paragraph will be added in the DESCRIPTION before the current paragraph that says: "The *strerror*( ) function shall not change the setting of *errno* if successful.":

MCL The *strerror_l*( ) function shall not change the setting of *errno* if successful.

The following will be added before the last paragraph in the DESCRIPTION:

MCL The *strerror_l*( ) function shall map the error number in *errnum* to a locale-dependent error message string in the locale represented by *locale* and shall return a pointer to it.

Change the following paragraph in the DESCRIPTION from:

The string pointed to shall not be modified by the application, but may be overwritten by a subsequent call to *strerror*( ) or *perror*( ).

to the following two paragraphs:

The string pointed to shall not be modified by the application. The string may be overwritten by
CX a subsequent call to *strerror*( ) or *perror*( ).

MCL The string may be overwritten by a subsquent call to *strerror_l*( ) in the same thread.

The following paragraph will be added to the RETURN VALUE section before the current last paragraph:

MCL Upon successful completion, *strerror_l*( ) shall return a pointer to the generated message string. If *errnum* is not a valid error number, *errno* may be set to [EINVAL], but a pointer to a message string shall still be returned. If any other error occurs, *errno* shall be set to indicate the error and a null pointer shall be returned.

The following will be added to the ERRORS section before the *strerror_r*( ) ‘‘may fail’’ entries:

The *strerror_l*( ) function may fail if:

MCL [EINVAL] The *locale* argument is not a valid locale object handle.

The following will be added to the RATIONALE:

The *strerror_l*( ) function is required to be thread-safe, thereby eliminating the need for an equivalent to the *strerror_r*( ) function.

3.2 New Interfaces

The following are new functions to add basic locale handling. These functions create, modify, duplicate, and release locale objects.

NAME

duplocale — duplicate a locale object

SYNOPSIS

MCL
```
#include <locale.h>

locale_t duplocale(locale_t locobj);
```

DESCRIPTION

The *duplocale*( ) function shall create a duplicate copy of the locale object referenced by the *locobj* argument.

RETURN VALUE

Upon successful completion, the *duplocale*( ) function shall return a handle for a new locale object. Otherwise, *duplocale*( ) shall return (**locale_t**)0 and set *errno* to indicate the error.

ERRORS

The *duplocale*( ) function shall fail if:

[ENOMEM] There is not enough memory available to create the locale object or load the locale data.

The *duplocale*( ) function may fail if:

[EINVAL] *locobj* is not a handle for a locale object.

EXAMPLES

Constructing an Altered Version of an Existing Locale Object

The following example shows a code fragment to create a slightly altered version of an existing locale object. The function takes a locale object and a locale name and it replaces the *LC_TIME* category data in the locale object with that from the named locale.

```
#include <locale.h>
...

locale_t
with_changed_lc_time (locale_t obj, const char *name)
{

    locale_t retval = duplocale (obj);
    if (retval != (locale_t) 0)
    {
        locale_t changed = newlocale (LC_TIME_MASK, name, retval);
        if (changed == (locale_t) 0)
            /* An error occurred. Free all allocated resources. */
            freelocale (retval);
        retval = changed;
    }
    return retval; }
}
```

APPLICATION USAGE

The *duplocale*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

The use of the *duplocale*( ) function is recommended for situations where a locale object is being used in multiple places, and it is possible that the lifetime of the locale object might end before all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is needed. This can be achieved by a call to *newlocale*( ) following the *duplocale*( ) call.

As with the *newlocale*( ) function, handles for locale objects created by the *duplocale*( ) function should be released by a corresponding call to *freelocale*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

freelocale( ), *newlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

freelocale — free resources allocated for a locale object

SYNOPSIS

MCL
```
#include <locale.h>

void freelocale(locale_t locobj);
```

DESCRIPTION

The *freelocale*( ) function shall cause the resources allocated for a locale object returned by a call to the *newlocale*( ) or *duplocale*( ) functions to be released.

Any use of a locale object that has been freed results in undefined behavior.

RETURN VALUE

None.

ERRORS

None.

EXAMPLES

Freeing Up a Locale Object

The following example shows a code fragment to free a locale object created by *newlocale*( ):

```
#include <locale.h>
...
/* Every locale object allocated with newlocale() should be
 * freed using freelocale():
 */
locale_t loc;
/* Get the locale. */
loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
/* ... Use the locale object ... */
...
/* Free the locale object resources. */
freelocale (loc);
```

APPLICATION USAGE

The *freelocale*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *newlocale*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

newlocale — create or modify a locale object

SYNOPSIS

MCL
```
#include <locale.h>

locale_t newlocale(int category_mask, const char *locale,
     locale_t base);
```

DESCRIPTION

The *newlocale*( ) function shall create a new locale object or modify an existing one. If the *base* argument is (**locale_t**)0, a new locale object shall be created. It is unspecified whether the locale object pointed to by *base* shall be modified or freed and a new locale object created.

The *category_mask* argument specifies the locale categories to be set or modified. Values for *category_mask* shall be constructed by a bitwise-inclusive OR of the symbolic constants *LC_CTYPE_MASK*, *LC_NUMERIC_MASK*, *LC_TIME_MASK*, *LC_COLLATE_MASK*, *LC_MONETARY_MASK*, and *LC_MESSAGES_MASK*, or any of the other implementation-defined *LC_*_MASK* values defined in **<locale.h>**.

For each category with the corresponding bit set in *category_mask* the data from the locale named by *locale* shall be used. In the case of modifying an existing locale object, the data from the locale named by *locale* shall replace the existing data within the locale object. If a completely new locale object is created, the data for all sections not requested by *category_mask* shall be taken from the default locale.

The following preset values of *locale* are defined for all settings of *category_mask*:

"POSIX"	Specifies the minimal environment for C-language translation called the POSIX locale.
"C"	Equivalent to `"POSIX"`.
" "	Specifies an implementation-defined native environment. This corresponds to the value of the associated environment variables, *LC_** and *LANG*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale and the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

If the *base* argument is not (**locale_t**)0 and the *newlocale*( ) function call succeeds, the contents of *base* are unspecified. Applications shall ensure that they stop using *base* as a locale object before calling *newlocale*( ). If the function call fails and the *base* argument is not (**locale_t**)0, the contents of *base* shall remain valid and unchanged.

The results are undefined if the *base* argument is the special locale object *LC_GLOBAL_LOCALE*.

RETURN VALUE

Upon successful completion, the *newlocale*( ) function shall return a handle which the caller may use on subsequent calls to *duplocale*( ), *freelocale*( ), and other functions taking a **locale_t** argument.

Upon failure, the *newlocale*( ) function shall return (**locale_t**)0 and set *errno* to indicate the error.

ERRORS

The *newlocale*( ) function shall fail if:

[ENOMEM]	There is not enough memory available to create the locale object or load the locale data.

[EINVAL]	The *category_mask* contains a bit that does not correspond to a valid category.
[ENOENT]	For any of the categories in *category_mask*, the locale data is not available.

The *newlocale*( ) function may fail if:

[EINVAL]	The *locale* argument is not a valid string pointer.

EXAMPLES

Constructing a Locale Object from Different Locales

The following example shows the construction of a locale where the *LC_CTYPE* category data comes from a locale *loc1* and the *LC_TIME* category data from a locale *tok2*:

```
#include <locale.h>
...
locale_t loc, new_loc;

/* Get the "loc1" data. */

loc = newlocale (LC_CTYPE_MASK, "loc1", NULL);
if (loc == (locale_t) 0)
    abort ();

/* Get the "loc2" data. */

new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
if (new_loc != (locale_t) 0)
    /* We don t abort if this fails. In this case this
       simply used to unchanged locale object. */
    loc = new_loc;

...
```

Freeing up a Locale Object

The following example shows a code fragment to free a locale object created by *newlocale*( ):

```
#include <locale.h>
...

/* Every locale object allocated with newlocale() should be
 * freed using freelocale():
 */

locale_t loc;

/* Get the locale. */

loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);

/* ... Use the locale object ... */
...

/* Free the locale object resources. */
freelocale (loc);
```

APPLICATION USAGE

The *newlocale*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

Handles for locale objects created by the *newlocale*( ) function should be released by a corresponding call to *freelocale*( ).

The special locale object *LC_GLOBAL_LOCALE* must not be passed for the *base* argument, even when returned by the *uselocale*( ) function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *freelocale*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

uselocale — use locale in current thread

SYNOPSIS

MCL THR
```
#include <locale.h>

locale_t uselocale(locale_t newloc);
```

DESCRIPTION

The *uselocale*( ) function shall set the current locale for the current thread to the locale represented by *newloc*.

The value for the *newloc* argument shall be one of the following:

1. A value returned by the *newlocale*( ) or *duplocale*( ) functions
2. The special locale object descriptor *LC_GLOBAL_LOCALE*
3. (**locale_t**)0

Once the *uselocale*( ) function has been called to install a thread-local locale, the behavior of every interface using data from the current locale shall be affected for the calling thread. The current locale for other threads shall remain unchanged.

If the *newloc* argument is a null pointer, the object returned is the current locale or *LC_GLOBAL_LOCALE* if there has been no previous call to *uselocale*( ) for the current thread.

If the *newloc* argument is *LC_GLOBAL_LOCALE*, the thread shall use the global locale determined by the *setlocale*( ) function.

RETURN VALUE

The *uselocale*( ) function returns the locale handle from the previous call for the current thread. If there was no such previous call, the function shall return the value *LC_GLOBAL_LOCALE*.

ERRORS

The *uselocale*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object.

EXAMPLES

None.

APPLICATION USAGE

The *uselocale*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

Unlike the *setlocale*( ) function, the *uselocale*( ) function does not allow replacing some locale categories only. Applications that need to install a locale which differs only in a few categories must use *newlocale*( ) to change a locale object equivalent to the currently used locale and install it.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *newlocale*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

3.3 Alternative Locale Versions of Existing Interfaces

The following functions are similar to existing standard functions. All of them take a locale object argument that specifies an alternative locale to be used instead of the process' or thread's current locale.

These references pages are to be merged into the System Interfaces volume of IEEE Std 1003.1-2001, Chapter 3, System Interfaces in alphabetic order.

NAME

isalnum_l — test for an alphanumeric character

SYNOPSIS

MCL
```
#include <ctype.h>

int isalnum_l(int c, locale_t locale);
```

DESCRIPTION

The *isalnum_l*( ) function shall test whether *c* is a character of class **alpha** or **digit** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isalnum_l*( ) function shall return non-zero if *c* is an alphanumeric character; otherwise, it shall return 0.

ERRORS

The *isalnum_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isalnum_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalpha_l( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue X.

NAME

isalpha_l — test for an alphabetic character

SYNOPSIS

MCL
```
#include <ctype.h>

int isalpha_l(int c, locale_t locale);
```

DESCRIPTION

The *isalpha_l*( ) function shall test whether *c* is a character of class **alpha** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isalpha_l*( ) function shall return non-zero if *c* is an alphabetic character; otherwise, it shall return 0.

ERRORS

The *isalpha_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isalpha_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue X.

NAME

isblank_l — test for a blank character

SYNOPSIS

MCL
```
#include <ctype.h>

int isblank_l(int c, locale_t locale);
```

DESCRIPTION

The *isblank_l*( ) function shall test whether *c* is a character of class **blank** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is a type **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isblank_l*( ) function shall return non-zero if *c* is a <blank>; otherwise, it shall return 0.

ERRORS

The *isblank_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isblank_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iscntrl_l — test for a control character

SYNOPSIS

MCL
```
#include <ctype.h>

int iscntrl_l(int c, locale_t locale);
```

DESCRIPTION

The *iscntrl_l*( ) function shall test whether *c* is a character of class **cntrl** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is a type **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iscntrl_l*( ) function shall return non-zero if *c* is a control character; otherwise, it shall return 0.

ERRORS

The *iscntrl_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iscntrl_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

isdigit_l — test for a decimal digit

SYNOPSIS

MCL

```
#include <ctype.h>

int isdigit_l(int c, locale_t locale);
```

DESCRIPTION

The *isdigit_l*( ) function shall test whether *c* is a character of class **digit** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isdigit_l*( ) function shall return non-zero if *c* is a decimal digit; otherwise, it shall return 0.

ERRORS

The *isdigit_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isdigit_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

isgraph_l — test for a visible character

SYNOPSIS

MCL
```
#include <ctype.h>

int isgraph_l(int c, locale_t locale);
```

DESCRIPTION

The *isgraph_l*( ) function shall test whether *c* is a character of class **graph** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isgraph_l*( ) function shall return non-zero if *c* is a character with a visible representation; otherwise, it shall return 0.

ERRORS

The *isgraph_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isgraph_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isdigit*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

islower_l — test for a lowercase letter

SYNOPSIS

MCL

```
#include <ctype.h>

int islower_l(int c, locale_t locale);
```

DESCRIPTION

The *islower_l*( ) function shall test whether *c* is a character of class **lower** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *islower_l*( ) function shall return non-zero if *c* is a lowercase letter; otherwise, it shall return 0.

ERRORS

The *islower_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

Testing for a Lowercase Letter

The following example tests whether the value is a lowercase letter, based on the locale of the user, then uses it as part of a key value.

```
#include <ctype.h>
#include <stdlib.h>
#include <locale.h>
...
char *keystr;
int elementlen, len;
char c;
...
locale_t loc = newlocale (LC_ALL_MASK, "", (locale_t) 0);
...
len = 0;
while (len < elementlen) {
    c = (char) (rand() % 256);
...
    if (islower_l(c, loc))
        keystr[len++] = c;
    }
...
```

APPLICATION USAGE

The *islower_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

isprint_l — test for a printable character

SYNOPSIS

MCL
```
#include <ctype.h>

int isprint_l(int c, locale_t locale);
```

DESCRIPTION

The *isprint_l*( ) function shall test whether *c* is a character of class **print** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isprint_l*( ) function shall return non-zero if *c* is a printable character; otherwise, it shall return 0.

ERRORS

The *isprint_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isprint_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

ispunct_l — test for a punctuation character

SYNOPSIS

MCL
```
#include <ctype.h>

int ispunct_l(int c, locale_t locale);
```

DESCRIPTION

The *ispunct_l*( ) function shall test whether *c* is a character of class **punct** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *ispunct_l*( ) function shall return non-zero if *c* is a punctuation character; otherwise, it shall return 0.

ERRORS

The *ispunct_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *ispunct_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *isspace_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

isspace_l — test for a white-space character

SYNOPSIS

MCL
```
#include <ctype.h>

int isspace_l(int c, locale_t locale);
```

DESCRIPTION

The *isspace_l*( ) function shall test whether *c* is a character of class **space** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isspace_l*( ) function shall return non-zero if *c* is a white-space character; otherwise, it shall return 0.

ERRORS

The *isspace_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isspace_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isupper_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

isupper_l — test for an uppercase letter

SYNOPSIS

MCL

```
#include <ctype.h>

int isupper_l(int c, locale_t locale);
```

DESCRIPTION

The *isupper_l*( ) function shall test whether *c* is a character of class **upper** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isupper_l*( ) function shall return non-zero if *c* is an uppercase letter; otherwise, it shall return 0.

ERRORS

The *isupper_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isupper_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswalnum_l — test for an alphanumeric wide-character code

SYNOPSIS

MCL

```
#include <wctype.h>

int iswalnum_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswalnum_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **alpha** or **digit** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswalnum_l*( ) function shall return non-zero if *wc* is an alphanumeric wide-character code; otherwise, it shall return 0.

ERRORS

The *iswalnum_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswalnum_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalpha_l( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<stdio.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswalpha_l — test for an alphabetic wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswalpha_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswalpha_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **alpha** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswalpha_l*( ) function shall return non-zero if *wc* is an alphabetic wide-character code; otherwise, it shall return 0.

ERRORS

The *iswalpha_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswalpha_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<stdio.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswblank_l — test for a blank wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswblank_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswblank_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **blank** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswblank_l*( ) function shall return non-zero if *wc* is a blank wide-character code; otherwise, it shall return 0.

ERRORS

The *iswblank_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswblank_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<stdio.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswcntrl_l — test for a control wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswcntrl_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswcntrl_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **cntrl** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswcntrl_l*( ) function shall return non-zero if *wc* is a control wide-character code; otherwise, it shall return 0.

ERRORS

The *iswcntrl_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswcntrl_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswctype_l — test character for a specified class

SYNOPSIS

MCL
```
#include <wctype.h>

int iswctype_l(wint_t wc, wctype_t charclass,
    locale_t locale);
```

DESCRIPTION

The *iswctype_l*( ) function shall determine whether the wide-character code *wc* has the character class *charclass*, returning true or false. The *iswctype_l*( ) function is defined on WEOF and wide-character codes corresponding to the valid character encodings in the locale represented by *locale*. If the *wc* argument is not in the domain of the function, the result is undefined. If the value of *charclass* is invalid (that is, not obtained by a call to *wctype*( )) the result is unspecified.

RETURN VALUE

The *iswctype_l*( ) function shall return non-zero (true) if and only if *wc* has the property described by *charclass*. If *charclass* is 0, *iswctype_l*( ) shall return 0.

ERRORS

The *iswctype_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswctype_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

The twelve strings `"alnum"`, `"alpha"`, `"blank"`, `"cntrl"`, `"digit"`, `"graph"`, `"lower"`, `"print"`, `"punct"`, `"space"`, `"upper"`, and `"xdigit"` are reserved for the standard character classes. In the table below, the functions in the left column are equivalent to the functions in the right column.

```
iswalnum_l(wc, locale)    iswctype_l(wc, wctype("alnum"), locale)
iswalpha_l(wc, locale)    iswctype_l(wc, wctype("alpha"), locale)
iswblank(wc, locale)    iswctype_l(wc, wctype("blank"), locale)
iswcntrl_l(wc, locale)    iswctype_l(wc, wctype("cntrl"), locale)
iswdigit_l(wc, locale)    iswctype_l(wc, wctype("digit"), locale)
iswgraph_l(wc, locale)    iswctype_l(wc, wctype("graph"), locale)
iswlower_l(wc, locale)    iswctype_l(wc, wctype("lower"), locale)
iswprint_l(wc, locale)    iswctype_l(wc, wctype("print"), locale)
iswpunct_l(wc, locale)    iswctype_l(wc, wctype("punct"), locale)
isspace_l(wc, locale)    iswctype_l(wc, wctype("space"), locale)
iswupper_l(wc, locale)    iswctype_l(wc, wctype("upper"), locale)
iswxdigit(wc, locale)    iswctype_l(wc, wctype("xdigit"), locale)
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), *wctype*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswdigit_l — test for a decimal digit wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswdigit_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswdigit_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **digit** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswdigit_l*( ) function shall return non-zero if *wc* is a decimal digit wide-character code; otherwise, it shall return 0.

ERRORS

The *iswdigit_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswdigit_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswgraph_l — test for a visible wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswgraph_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswgraph_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **graph** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswgraph_l*( ) function shall return non-zero if *wc* is a wide-character code with a visible representation; otherwise, it shall return 0.

ERRORS

The *iswgraph_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswgraph_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswlower_l — test for a lowercase letter wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswlower_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswlower_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **lower** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswlower_l*( ) function shall return non-zero if *wc* is a lowercase letter wide-character code; otherwise, it shall return 0.

ERRORS

The *iswlower_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswlower_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswprint_l — test for a printable wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswprint_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswprint_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **print** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswprint_l*( ) function shall return non-zero if *wc* is a printable wide-character code; otherwise, it shall return 0.

ERRORS

The *iswprint_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswprint_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswpunct_l — test for a punctuation wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswpunct_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswpunct_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **punct** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswpunct_l*( ) function shall return non-zero if *wc* is a punctuation wide-character code; otherwise, it shall return 0.

ERRORS

The *iswpunct_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswpunct_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *isspace_l*( ), *iswupper_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswspace_l — test for a white-space wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswspace_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswspace_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **space** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswspace_l*( ) function shall return non-zero if *wc* is a white-space wide-character code; otherwise, it shall return 0.

ERRORS

The *iswspace_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswspace_l*( ) function is part of the Multiple Concurrent Locales option be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *iswupper_l*( ), *iswxdigit_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswupper_l — test for an uppercase letter wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswupper_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswupper_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **upper** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswupper_l*( ) function shall return non-zero if *wc* is an uppercase letter wide-character code; otherwise, it shall return 0.

ERRORS

The *iswupper_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswupper_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswxdigit*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

iswxdigit_l — test for a hexadecimal digit wide-character code

SYNOPSIS

MCL
```
#include <wctype.h>

int iswxdigit_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *iswxdigit_l*( ) function shall test whether *wc* is a wide-character code representing a character of class **xdigit** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswxdigit_l*( ) function shall return non-zero if *wc* is a hexadecimal digit wide-character code; otherwise, it shall return 0.

ERRORS

The *iswxdigit_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *iswxdigit_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum_l( ), *iswalpha_l*( ), *iswcntrl_l*( ), *iswctype_l*( ), *iswdigit_l*( ), *iswgraph_l*( ), *iswlower_l*( ), *iswprint_l*( ), *iswpunct_l*( ), *isspace_l*( ), *iswupper_l*( ), *uselocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

isxdigit_l — test for a hexadecimal digit

SYNOPSIS

MCL
```
#include <ctype.h>

int isxdigit_l(int c, locale_t locale);
```

DESCRIPTION

The *isxdigit_l*( ) function shall test whether *c* is a character of class **xdigit** in the locale represented by *locale*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isxdigit_l*( ) function shall return non-zero if *c* is a hexadecimal digit; otherwise, it shall return 0.

ERRORS

The *isxdigit_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *isxdigit_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum_l( ), *isalpha_l*( ), *iscntrl_l*( ), *isdigit*( ), *isgraph_l*( ), *islower_l*( ), *isprint_l*( ), *ispunct_l*( ), *isspace_l*( ), *isupper_l*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

nl_langinfo_l — language information

SYNOPSIS

MCL
```
#include <langinfo.h>

char *nl_langinfo_l(nl_item item, locale_t locale);
```

DESCRIPTION

The *nl_langinfo_l*( ) function shall return a pointer to a string containing information relevant to the particular language or cultural area defined in the locale represented to by *locale* (see **<langinfo.h>**). The manifest constant names and values of *item* are defined in **<langinfo.h>**. For example:

```
nl_langinfo_l(ABDAY_1, loc)
```

would return a pointer to the string `"Dom"` if the identified language of the locale represented by *loc* was Portuguese, and `"Sun"` if the identified language of the locale represented by *loc* was English.

RETURN VALUE

In a locale where *langinfo* data is not defined, *nl_langinfo_l*( ) shall return a pointer to the corresponding string in the POSIX locale. In all locales, *nl_langinfo_l*( ) shall return a pointer to an empty string if *item* contains an invalid setting.

This pointer may point to static data that may be overwritten on the next call.

ERRORS

The *nl_langinfo_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *nl_langinfo_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

The array pointed to by the return value should not be modified by the program, but may be modified by further calls to *nl_langinfo_l*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

uselocale( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<langinfo.h>**, **<locale.h>**, **<nl_types.h>**

CHANGE HISTORY

First released in Issue X.

NAME

strcasecmp_l, strncasecmp_l — case-insensitive string comparisons

SYNOPSIS

MCL
```
#include <strings.h>
```

```
int strcasecmp_l(const char *s1, const char *s2,
    locale_t locale);
int strncasecmp_l(const char *s1, const char *s2,
    size_t n, locale_t locale);
```

DESCRIPTION

The *strcasecmp_l*( ) function shall compare, while ignoring differences in case, the string pointed to by *s1* to the string pointed to by *s2*. The *strncasecmp_l*( ) function shall compare, while ignoring differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

The information about the case of the characters come from the locale represented by *locale*.

RETURN VALUE

Upon completion, *strcasecmp_l*( ) shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to by *s2*, respectively.

Upon successful completion, *strncasecmp_l*( ) shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

ERRORS

The *strcasecmp_l*( ) and *strncasecmp_l*( ) functions may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *strcasecmp_l*( ) and *strncasecmp_l*( ) functions are part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**

CHANGE HISTORY

First released in Issue X.

NAME

strcoll_l — string comparison using collating information

SYNOPSIS

MCL
```
#include <string.h>

int strcoll_l(const char *s1, const char *s2,
    locale_t locale);
```

DESCRIPTION

The *strcoll_l*( ) function shall compare the string pointed to by *s1* to the string pointed to by *s2*, both interpreted as appropriate to the *LC_COLLATE* category of the locale represented by *locale*.

The *strcoll_l*( ) function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strcoll_l*( ), then check *errno*.

RETURN VALUE

Upon successful completion, *strcoll_l*( ) shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2* when both are interpreted as appropriate to the locale represented by *locale*. On error, *strcoll_l*( ) may set *errno*, but no return value is reserved to indicate an error.

ERRORS

The *strcoll_l*( ) function may fail if:

[EINVAL]	The *s1* or *s2* arguments contain characters outside the domain of the collating sequence.
[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *strcoll_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

The *strxfrm_l*( ) and *strcmp*( ) functions should be used for sorting large lists.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcmp( ), *strxfrm*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue X.

NAME

strfmon_l — convert monetary value to a string

SYNOPSIS

MCL

```
#include <monetary.h>

ssize_t strfmon_l(char *restrict s, size_t maxsize,
    locale_t locale, const char *restrict format, ...);
```

DESCRIPTION

The *strfmon_l*( ) function shall be equivalent to the *strfmon*( ) function, except that the current locale data used is from the locale represented by *locale*.

RETURN VALUE

See *strfmon*( ).

ERRORS

See *strfmon*( ), with the additional error below.

The *strfmon_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object.

EXAMPLES

None.

APPLICATION USAGE

The *strfmon_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

SEE ALSO

fprintf( ), *localeconv*( ), *strfmon*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<monetary.h>**

CHANGE HISTORY

First released in Issue X.

NAME

strftime_l — convert date and time to a string

SYNOPSIS

MCL

```
#include <time.h>

size_t strftime_l(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr,
    locale_t locale);
```

DESCRIPTION

The *strftime_l*( ) function shall be equivalent to the *strftime*( ) function, except that the current locale data used is from the locale represented by *locale*.

RETURN VALUE

See *strftime*( ).

ERRORS

The *strftime_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *strftime_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *getdate*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *tzset*( ), *uselocale*( ), *utime*( ), Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5, LC_TIME, **<time.h>**

CHANGE HISTORY

First released in Issue X.

NAME

strxfrm_l — string transformation

SYNOPSIS

MCL

```
#include <string.h>

size_t strxfrm_l(char *restrict s1, const char *restrict s2,
    size_t n, locale_t locale);
```

DESCRIPTION

The *strxfrm_l*( ) function shall transform the string pointed to by *s2* and place the resulting string into the array pointed to by *s1*. The transformation is such that if *strcmp*( ) is applied to two transformed strings, it shall return a value greater than, equal to, or less than 0, corresponding to the result of *strcoll_l*( ) applied to the same two original strings with the same *locale*. No more than *n* bytes are placed into the resulting array pointed to by *s1*, including the terminating null byte. If *n* is 0, *s1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

The *strxfrm_l*( ) function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strxfrm_l*( ), then check *errno*.

RETURN VALUE

Upon successful completion, *strxfrm_l*( ) shall return the length of the transformed string (not including the terminating null byte). If the value returned is *n* or more, the contents of the array pointed to by *s1* are unspecified.

On error, *strxfrm_l*( ) may set *errno* but no return value is reserved to indicate an error.

ERRORS

The *strxfrm_l*( ) function may fail if:

[EINVAL]	The string pointed to by the *s2* argument contains characters outside the domain of the collating sequence.
[EINVAL]	*locale* is not a valid locale object.

EXAMPLES

None.

APPLICATION USAGE

The *strxfrm_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcmp( ), *strcoll*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue X.

NAME

tolower_l — transliterate uppercase characters to lowercase

SYNOPSIS

MCL

```
#include <ctype.h>

int tolower_l(int c, locale_t locale);
```

DESCRIPTION

The *tolower_l*( ) function has as a domain a type **int**, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined. If the argument of *tolower_l*( ) represents an uppercase letter, and there exists a corresponding lowercase letter (as defined by character type information in the category *LC_CTYPE* in the locale represented by *locale*), the result shall be the corresponding lowercase letter. All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, *tolower_l*( ) shall return the lowercase letter corresponding to the argument passed; otherwise, it shall return the argument unchanged.

ERRORS

The *tolower_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *tolower_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

uselocale( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

toupper_l — transliterate lowercase characters to uppercase

SYNOPSIS

MCL

```
#include <ctype.h>

int toupper_l(int c, locale_t locale);
```

DESCRIPTION

The *toupper_l*( ) function has as a domain a type **int**, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined. If the argument of *toupper_l*( ) represents a lowercase letter, and there exists a corresponding uppercase letter (as defined by character type information in the category *LC_CTYPE* in the locale represented by *locale*), the result shall be the corresponding uppercase letter. All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, *toupper_l*( ) shall return the uppercase letter corresponding to the argument passed.

ERRORS

The *toupper_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *toupper_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

uselocale( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue X.

NAME

towctrans_l — wide-character transliteration

SYNOPSIS

MCL
```
#include <wctype.h>

wint_t towctrans_l(wint_t wc, wctrans_t desc,
    locale_t locale);
```

DESCRIPTION

The *towctrans_l*( ) function shall transliterate the wide-character code *wc* using the mapping described by *desc*. The setting of the *LC_CTYPE* category in the locale represented by *locale* should be the same as during the call to *wctrans_l*( ) that returned the value *desc*. If the value of *desc* is invalid (that is, not obtained by a call to *wctrans_l*( ) with the same locale object *locale*) the result is unspecified.

An application wishing to check for error situations should set *errno* to 0 before calling *towctrans_l*( ). If *errno* is non-zero on return, an error has occurred.

RETURN VALUE

If successful, the *towctrans_l*( ) function shall return the mapped value of *wc* using the mapping described by *desc*. Otherwise, it shall return *wc* unchanged.

ERRORS

The *towctrans_l*( ) function may fail if:

[EINVAL]	*desc* contains an invalid transliteration descriptor.
[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *towctrans_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

The strings `"tolower"` and `"toupper"` are reserved for the standard mapping names. In the table below, the functions in the left column are equivalent to the functions in the right column.

```
towlower_l(wc, locale)     towctrans_l(wc, wctrans("tolower"), locale)
towupper_l(wc, locale)     towctrans_l(wc, wctrans("toupper"), locale)
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

towlower( ), *towupper*( ), *wctrans*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

towlower_l — transliterate uppercase wide-character code to lowercase

SYNOPSIS

MCL
```
#include <wctype.h>

wint_t towlower_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *towlower_l*( ) function has as a domain a type **wint_t**, the value of which the application shall ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a valid character in the current locale or the value of WEOF. If the argument has any other value, the behavior is undefined. If the argument of *towlower_l*( ) represents an uppercase wide-character code, and there exists a corresponding lowercase wide-character code (as defined by character type information in the locale category *LC_CTYPE* in the locale represented by *locale*), the result shall be the corresponding lowercase wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, *towlower_l*( ) shall return the lowercase letter corresponding to the argument passed; otherwise, it shall return the argument unchanged.

ERRORS

The *towlower_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *towlower_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

uselocale( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wctype.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

towupper_l — transliterate lowercase wide-character code to uppercase

SYNOPSIS

MCL

```
#include <wctype.h>

wint_t towupper_l(wint_t wc, locale_t locale);
```

DESCRIPTION

The *towupper_l*( ) function has as a domain a type **wint_t**, the value of which the application shall ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a valid character in the current locale or the value of WEOF. If the argument has any other value, the behavior is undefined. If the argument of *towupper_l*( ) represents a lowercase wide-character code, and there exists a corresponding uppercase wide-character code (as defined by character type information in the locale category *LC_CTYPE* in the locale represented by *locale*), the result shall be the corresponding uppercase wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, *towupper_l*( ) shall return the uppercase letter corresponding to the argument passed. Otherwise, it shall return the argument unchanged.

ERRORS

The *towupper_l*( ) function may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *towupper_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

uselocale( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<locale.h>**, **<wctype.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcscasecmp_l, wcsncasecmp_l — case-insensitive wide-character string comparisons

SYNOPSIS

MCL

```
#include <wchar.h>

int wcscasecmp_l(const char *ws1, const char *ws2,
    locale_t locale);
int wcsncasecmp_l(const char *ws1, const char *ws2,
    size_t n, locale_t locale);
```

DESCRIPTION

The *wcscasecmp_l*( ) function shall compare, while ignoring differences in case, the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*. The *wcsncasecmp_l*( ) function shall compare, while ignoring differences in case, not more than *n* wide-characters from the string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The information about the case of the characters come from the locale represented by *locale*.

RETURN VALUE

Upon completion, *wcscasecmp_l*( ) shall return an integer greater than, equal to, or less than 0, if the wide-character string pointed to by *ws1* is, ignoring case, greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

Upon successful completion, *wcsncasecmp_l*( ) shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *ws1* is, ignoring case, greater than, equal to, or less than the possibly null-terminated array pointed to by *ws2*, respectively.

ERRORS

The *wcscasecmp_l*( ) and *wcsncasecmp_l*( ) functions may fail if:

[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *wcscasecmp_l*( ) and *wcsncasecmp_l*( ) functions are part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcscoll_l — wide-character string comparison using collating information

SYNOPSIS

MCL
```
#include <wchar.h>
```

```
int wcscoll_l(const wchar_t *ws1, const wchar_t *ws2,
    locale_t locale);
```

DESCRIPTION

The *wcscoll_l*( ) function shall compare the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*, both interpreted as appropriate to the *LC_COLLATE* category of the locale represented by *locale*.

The *wcscoll_l*( ) function shall not change the setting of *errno* if successful.

An application wishing to check for error situations should set *errno* to 0 before calling *wcscoll_l*( ). If *errno* is non-zero on return, an error has occurred.

RETURN VALUE

Upon successful completion, *wcscoll_l*( ) shall return an integer greater than, equal to, or less than 0, according to whether the wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character string pointed to by *ws2*, when both are interpreted as appropriate to the locale represented by *locale*. On error, *wcscoll_l*( ) shall set *errno*, but no return value is reserved to indicate an error.

ERRORS

The *wcscoll_l*( ) function may fail if:

[EINVAL]	The *ws1* or *ws2* arguments contain wide-character codes outside the domain of the collating sequence.
[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *wcscoll_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscmp( ), *wcsxfrm*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcsxfrm_l — wide-character string transformation

SYNOPSIS

MCL
```
#include <wchar.h>

size_t wcsxfrm_l(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n, locale_t locale);
```

DESCRIPTION

The *wcsxfrm_l*( ) function shall transform the wide-character string pointed to by *ws2* and place the resulting wide-character string into the array pointed to by *ws1*. The transformation shall be such that if *wcscmp*( ) is applied to two transformed wide strings, it shall return a value greater than, equal to, or less than 0, corresponding to the result of *wcscoll*( ) applied to the same two original wide-character strings and the same locale object *locale*. No more than *n* wide-character codes shall be placed into the resulting array pointed to by *ws1*, including the terminating null wide-character code. If *n* is 0, *ws1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

The *wcsxfrm_l*( ) function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *wcsxfrm_l*( ), then check *errno*.

RETURN VALUE

The *wcsxfrm_l*( ) function shall return the length of the transformed wide-character string (not including the terminating null wide-character code). If the value returned is *n* or more, the contents of the array pointed to by *ws1* are unspecified.

On error, the *wcsxfrm_l*( ) function may set *errno*, but no return value is reserved to indicate an error.

ERRORS

The *wcsxfrm_l*( ) function may fail if:

[EINVAL]	The wide-character string pointed to by *ws2* contains wide-character codes outside the domain of the collating sequence.

locale is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *wcsxfrm_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscmp( ), *wcscoll*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wctrans_l — define character mapping

SYNOPSIS

MCL

```
#include <wctype.h>

wctrans_t wctrans_l(const char *charclass, locale_t locale);
```

DESCRIPTION

The *wctrans_l*( ) function is defined for valid character mapping names identified in the current locale. The *charclass* is a string identifying a generic character mapping name for which codeset-specific information is required. The following character mapping names are defined in all locales: **tolower** and **toupper**.

The function shall return a value of type **wctrans_t**, which can be used as the second argument to subsequent calls of *towctrans_l*( ). The *wctrans_l*( ) function shall determine values of **wctrans_t** according to the rules of the coded character set defined by character mapping information in the locale represented by *locale* (category *LC_CTYPE*). The values returned by *wctrans_l*( ) are only valid in calls to *wctrans_l*( ) with the same locale object *locale*.

RETURN VALUE

The *wctrans_l*( ) function shall return 0 and may set *errno* to indicate the error if the given character mapping name is not valid for the current locale (category *LC_CTYPE*); otherwise, it shall return a non-zero object of type **wctrans_t** that can be used in calls to *towctrans_l*( ).

ERRORS

The *wctrans_l*( ) function may fail if:

[EINVAL]	The character mapping name pointed to by *charclass* is not valid in the current locale.
[EINVAL]	*locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *wctrans_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

towctrans_l( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wctype_l — define character class

SYNOPSIS

MCL
```
#include <wctype.h>

wctype_t wctype_l(const char *property, locale_t locale);
```

DESCRIPTION

The *wctype_l*( ) function is defined for valid character class names as defined in the locale represented by *locale*. The *property* argument is a string identifying a generic character class for which codeset-specific type information is required. The following character class names shall be defined in all locales:

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

Additional character class names defined in the locale definition file (category *LC_CTYPE*) can also be specified.

The function shall return a value of type **wctype_t**, which can be used as the second argument to subsequent calls of *iswctype_l*( ). The *wctype_l*( ) function shall determine values of **wctype_t** according to the rules of the coded character set defined by character type information in the locale represented by *locale* (category *LC_CTYPE*). The values returned by *wctype_l*( ) are only valid in calls to *iswctype_l*( ) with the same *locale*.

RETURN VALUE

The *wctype_l*( ) function shall return 0 if the given character class name is not valid for the current locale (category *LC_CTYPE*); otherwise, it shall return an object of type **wctype_t** that can be used in calls to *iswctype_l*( ).

ERRORS

The *wctype_l*( ) function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The *wctype_l*( ) function is part of the Multiple Concurrent Locales option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswctype_l( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wctype.h>**

CHANGE HISTORY

First released in Issue X.

Index