X11R6 Sample Implementation Frame Work

Katsuhisa Yano
TOSHIBA Corporation
Yoshio Horiuchi
IBM Japan

X11R6 Sample Implementation Frame Work

by
Katsuhisa Yano
TOSHIBA Corporation
Yoshio Horiuchi
IBM Japan

X Version 11, Release 7

Copyright © 1994 TOSHIBA Corporation
Copyright © 1994 IBM Corporation

Permission to use, copy, modify, and distribute this documentation for any purpose and without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies. TOSHIBA Corporation and IBM Corporation make no representations about the suitability for any purpose of the information in this document. This documentation is provided as is without express or implied warranty.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of The Open Group shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from X Consortium.

X Window System is a trademark of The Open Group.

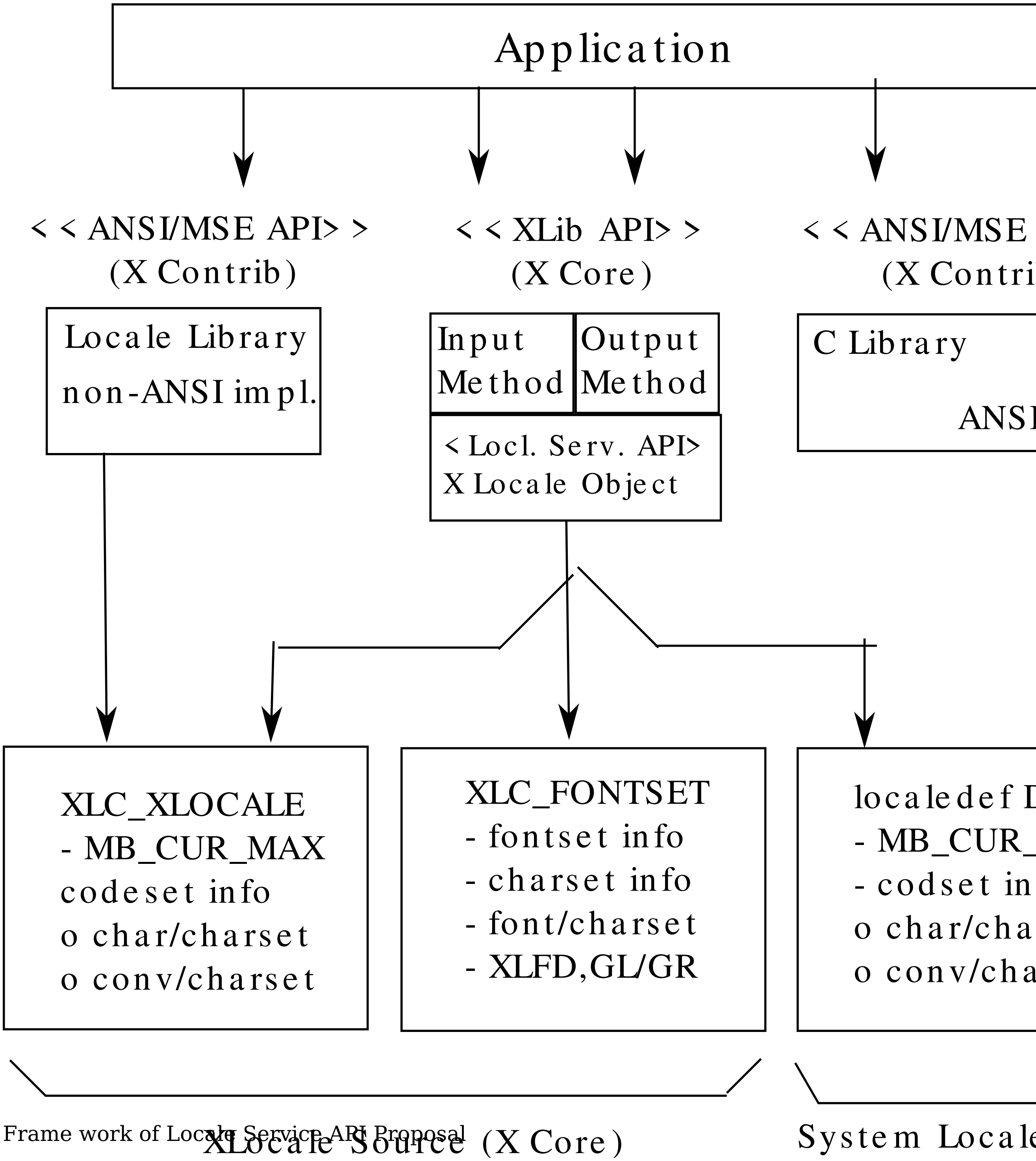
Table of Contents

Chapter 1. Framework

Preface

This document proposes to define the structures, methods and their signatures that are expected to be common to all locale dependent functions within the Xlib sample implementation. The following illustration (Fig.1) is proposed to outline the separating of the components within the sample implementation.

Preface drawing.



Frame work of Locale Service API Proposal

Generally speaking, the internationalized portion of Xlib (Locale Dependent X, LDX) consists of three objects; locale (LC) , input method (IM) and output method (OM). The LC provides a set of information that depends on user's language environment. The IM manages text inputing, and the OM manages text drawing. Both IM and OM highly depend on LC data.

In X11R5, there are two sample implementations, Ximp and Xsi, for Xlib internationalization. But in both implementations, IM and OM actually refer the private extension of LC. It breaks coexistence of these two sample implementations. For example, if a user creates a new OM for special purpose as a part of Ximp, it will not work with Xsi.

As a solution of this problem, we propose to define the standard APIs between these three objects, and define the structure that are common to these objects.

Objective

- Explain the current X11R6 sample implementation
- Document the common set of locale dependent interfaces
- Provide more flexible pluggable layer

Locale Object Binding Functions

This chapter describes functions related locale object binding for implementing the pluggable layer.

A locale loader is an entry point for locale object, which instantiates XLCd object and binds locale methods with specified locale name. The behavior of loader is implementation dependent. And, what kind of loaders are available is also implementation dependent.

The loader is called in `_XOpenLC`, but caller of `_XOpenLC` does not need to care about its inside. For example, if the loader is implemented with dynamic load functions, and the dynamic module is expected to be unloaded when the corresponding XLCd is freed, close methods of XLCdMethods should handle unloading.

Initializing a locale loader list

void _XlcInitLoader

The `_XlcInitLoader` function initializes the locale loader list with vendor specific manner. Each loader is registered with calling `_XlcAddLoader`. The number of loaders and their order in the loader list is implementation dependent.

Add a loader

```
typedef XLCd (*XLCdLoadProc)(name);
     char *name;

typedef int XlcPosition;

#define   XlcHead
#define   XlcTail

Bool  _XlcAddLoader( proc,  position);
```

The `_XlcAddLoader` function registers the specified locale loader "*proc*" to the internal loader list. The position specifies that the loader "*proc*" should be placed in the top of the loader list(XlcHead) or last(XlcTail).

The object loader is called from the top of the loader list in order, when calling time.

`Remove a loader`

`void` **`_XlcRemoveLoader`**`(` *`proc`*`);`

The `_XlcRemoveLoader` function removes the locale loader specified by "*proc*" from the loader list.

Current implementation provides following locale loaders;

```
_XlcDefaultLoader
_XlcGenericLoader
_XlcEucLoader
_XlcSjisLoader
_XlcUtfLoader
_XaixOsDynamicLoad
```

Locale Method Interface

This chapter describes the locale method API, which is a set of accessible functions from both IM and OM parts. The locale method API provides the functionalities; obtaining locale dependent information, handling charset, converting text, etc.

As a result of using these APIs instead of accessing vender private extension of the locale object, we can keep locale, IM and OM independently each other.

Locale Method Functions

`Open a Locale Method`

`XLCd` **`_XOpenLC`**`(` *`*name`*`);`

The `_XOpenLC` function opens a locale method which corresponds to the specified locale name. `_XOpenLC` calls a locale object loader, which is registered via `_XlcAddLoader into the internal loader list. If the called loader` is valid and successfully opens a locale, `_XOpenLC` returns the XLCd. If the loader is invalid or failed to open a locale, `_XOpenLC` calls the next loader. If all registered loaders cannot open a locale, `_XOpenLC` returns NULL.

XLCd _XlcCurrentLC

The `_XlcCurrentLC` function returns an XLCd that are bound to current locale.

Close a Locale Method

`void` **`_XCloseLC`**`(` *`lcd`*`);`

The `_XCloseLC` function close a locale method the specified lcd.

Obtain Locale Method values

`char *`**`_XGetLCValues`**`(` *`lcd`*`);`

The `_XGetLCValues` function returns NULL if no error occurred; otherwise, it returns the name of the first argument that could not be obtained. The following values are defined as standard arguments. Other values are implementation dependent.

Name	Type	Description
XlcNCodeset	char*	codeset part of locale name
XlcNDefaultString	char*	XDefaultString()
XlcNEncodingName	char*	encoding name
XlcNLanguage	char*	language part of locale name
XlcNMbCurMax	int	ANSI C MB_CUR_MAX
XlcNStateDependentEncoding	Bool	is state-dependent encoding or not
XlcNTerritory	char*	territory part of locale name

Charset functions

The XlcCharSet is an identifier which represents a subset of characters (character set) in the locale object.

```
typedef enum {
      XlcUnknown, XlcC0, XlcGL, XlcC1, XlcGR, XlcGLGR, XlcOther
} XlcSide;

typedef struct _XlcCharSetRec *XlcCharSet;

typedef struct {
      char *name;
      XPointer value;
} XlcArg, *XlcArgList;

typedef char* (*XlcGetCSValuesProc)(charset, args, num_args);
      XlcCharSet charset;
      XlcArgList args;
      int num_args;

typedef struct _XlcCharSetRec {
      char *name;
      XrmQuark xrm_name;
      char *encoding_name;
      XrmQuark xrm_encoding_name;
      XlcSide side;
      int char_size;
      int set_size;
      char *ct_sequence;
      XlcGetCSValuesProc get_values;
} XlcCharSetRec;
```

Get an XlcCharSet

```
XlcCharSet  _XlcGetCharSet( *name);
```

The `_XlcGetCharSet` function gets an XlcCharSet which corresponds to the charset name specified by "*name*". `_XlcGetCharSet` returns NULL, if no XlcCharSet bound to specified "*name*".

The following character sets are pre-registered.

Name	**Description**
ISO8859-1:GL	7-bit ASCII graphics (ANSI X3.4-1968), Left half of ISO 8859 sets
JISX0201.1976-0:GL	Left half of JIS X0201-1976 (reaffirmed 1984), 8-Bit Alphanumeric-Katakana Code
ISO8859-1:GR	Right half of ISO 8859-1, Latin alphabet No. 1
ISO8859-2:GR	Right half of ISO 8859-2, Latin alphabet No. 2
ISO8859-3:GR	Right half of ISO 8859-3, Latin alphabet No. 3
ISO8859-4:GR	Right half of ISO 8859-4, Latin alphabet No. 4
ISO8859-7:GR	Right half of ISO 8859-7, Latin/Greek alphabet
ISO8859-6:GR	Right half of ISO 8859-6, Latin/Arabic alphabet
ISO8859-8:GR	Right half of ISO 8859-8, Latin/Hebrew alphabet
ISO8859-5:GR	Right half of ISO 8859-5, Latin/Cyrillic alphabet
ISO8859-9:GR	Right half of ISO 8859-9, Latin alphabet No. 5
JISX0201.1976-0:GR	Right half of JIS X0201-1976 (reaffirmed 1984), 8-Bit Alphanumeric-Katakana Code
GB2312.1980-0:GL	GB2312-1980, China (PRC) Hanzi defined as GL
GB2312.1980-0:GR	GB2312-1980, China (PRC) Hanzi defined as GR
JISX0208.1983-0:GL	JIS X0208-1983, Japanese Graphic Character Set defined as GL
JISX0208.1983-0:GR	JIS X0208-1983, Japanese Graphic Character Set defined as GR
KSC5601.1987-0:GL	KS C5601-1987, Korean Graphic Character Set defined as GL
KSC5601.1987-0:GR	KS C5601-1987, Korean Graphic Character Set defined as GR
JISX0212.1990-0:GL	JIS X0212-1990, Japanese Graphic Character Set defined as GL
JISX0212.1990-0:GR	JIS X0212-1990, Japanese Graphic Character Set defined as GR

Add an XlcCharSet

```
Bool  _XlcAddCharSet( charset);
```

The `_XlcAddCharSet` function registers XlcCharSet specified by "*charset*".

`Obtain Character Set values`

```
char *  _XlcGetCSValues( charset,  ...);
```

The `_XlcGetCSValues` function returns NULL if no error occurred; otherwise, it returns the name of the first argument that could not be obtained. The following values are defined as standard arguments. Other values are implementation dependent.

Name	Type	Description
XlcNName	char*	charset name
XlcNEncodingName	char*	XLFD CharSet Registry and Encoding
XlcNSide	XlcSide	charset side (GL, GR, ...)
XlcNCharSize	int	number of octets per character
XlcNSetSize	int	number of character sets
XlcNControlSequence	char*	control sequence of Compound Text

Converter Functions

We provide a set of the common converter APIs, that are independent from both of source and destination text type.

```
typedef struct _XlcConvRec *XlcConv;

typedef void (*XlcCloseConverterProc)(conv);
      XlcConv conv;

typedef int (*XlcConvertProc)(conv, from, from_left, to, to_left, args, num_args);
      XlcConv conv;
      XPointer *from;
      int *from_left;
      XPointer *to;
      int *to_left;
      XPointer *args;
      int num_args;

typedef void (*XlcResetConverterProc)(conv);
      XlcConv conv;

typedef struct _XlcConvMethodsRec {
      XlcCloseConverterProc close;
      XlcConvertProc convert;
      XlcResetConverterProc reset;
} XlcConvMethodsRec, *XlcConvMethods;

typedef struct _XlcConvRec {
    XlcConvMethods methods;
    XPointer state;
} XlcConvRec;

Open a converter
```

`XlcConv **_XlcOpenConverter**( *from_lcd*, **from_type*, *to_lcd*, **to_type*);`

`_XlcOpenConverter` function opens the converter which converts a text from specified "*from_type*" to specified "*to_type*" encoding. If the function cannot find proper converter or cannot open a corresponding converter, it returns NULL. Otherwise, it returns the conversion descriptor.

The following types are pre-defined. Other types are implementation dependent.

Name	**Type**	**Description**	**Arguments**
XlcNMultiByte	char *	multibyte	-
XlcNWideChar	wchar_t *	wide character	-
XlcNCompoundText	char *	COMPOUND_TEXT	-
XlcNString	char *	STRING	-
XlcNCharSet	char *	per charset	XlcCharSet
XlcNChar	char *	per character	XlcCharSet

Close a converter

`void **_XlcCloseConverter**( *conv*);`

The `_XlcCloseConverter` function closes the specified converter "*conv*".

Code conversion

`int **_XlcConvert**( *conv*, **from*, **from_left*, **to*, **to_left*, **args*, *num_args*);`

The `_XlcConvert` function converts a sequence of characters from one type, in the array specified by "*from*", into a sequence of corresponding characters in another type, in the array specified by "*to*". The types are those specified in the `_XlcOpenConverter()` call that returned the conversion descriptor, "*conv*". The arguments "*from*", "*from_left*", "*to*" and "*to_left*" have the same specification of XPG4 iconv function.

For state-dependent encodings, the conversion descriptor "*conv*" is placed into its initial shift state by a call for which "*from*" is a NULL pointer, or for which "*from*" points to a null pointer.

The following 2 converters prepared by locale returns appropriate charset (XlcCharSet) in an area pointed by args[0].

From	**To**	**Description**
XlcNMultiByte	XlcNCharSet	Segmentation (Decomposing)
XlcNWideChar	XlcNCharSet	Segmentation (Decomposing)

The conversion, from XlcNMultiByte/XlcNWideChar to XlcNCharSet, extracts a segment which has same charset encoding characters. More than one segment cannot be converted in a call.

Reset a converter

`void **_XlcResetConverter**( *conv*);`

The `_XlcResetConverter` function reset the specified converter "*conv*".

Register a converter

```
typedef XlcConv (*XlcOpenConverterProc)(from_lcd, from_type, to_lcd, to_type);
      XLCd from_lcd;
      char *from_type;
      XLCd to_lcd;
      char *to_type;

Bool  _XlcSetConverter( from_lcd,  *from,  to_lcd,  *to,  converter);
```

The `XlcSetConverter` function registers a converter which convert from "*from_type*" to "*to_type*" into the converter list (in the specified XLCd).

X Locale Database functions

X Locale Database contains the subset of user's environment that depends on language. The following APIs are provided for accessing X Locale Database and other locale relative files.

For more detail about X Locale Database, please refer X Locale Database Definition document.

Get a resource from database

```
void  _XlcGetResource( lcd,  *category,  *class,  ***value,  *count);
```

The `_XlcGetResource` function obtains a locale dependent data which is associated with the locale of specified "*lcd*". The locale data is provided by system locale or by X Locale Database file, and what kind of data is available is implementation dependent.

The specified "*category*" and "*class*" are used for finding out the objective locale data.

The returned value is returned in value argument in string list form, and the returned count shows the number of strings in the value.

The returned value is owned by locale method, and should not be modified or freed by caller.

Get a locale relative file name

```
char *_XlcFileName( lcd,  *category);
```

The `_XlcFileName` functions returns a file name which is bound to the specified "*lcd*" and "*category*", as a null-terminated string. If no file name can be found, or there is no readable file for the found file name, `_XlcFileName` returns NULL. The returned file name should be freed by caller.

The rule for searching a file name is implementation dependent. In current implementation, `_XlcFileName` uses "{category}.dir" file as mapping table, which has pairs of strings, a full locale name and a corresponding file name.

Utility Functions

Compare Latin-1 strings

```
int  _XlcCompareISOLatin1( *str2);
```

```
int  _XlcNCompareISOLatin1( *str2,  len);
```

The `_XlcCompareIsoLatin1` function to compares two ISO-8859-1 strings. Bytes representing ASCII lower case letters are converted to upper case before making the comparison. The value returned is an integer less than, equal to, or greater than zero, depending on whether "*str1*" is lexicographicly less than, equal to, or greater than "*str2*".

The `_XlcNCompareIsoLatin1` function is identical to `_XlcCompareISOLatin1`, except that at most "*len*" bytes are compared.

Resource Utility

```
int  XlcNumber( array);
```

Similar to XtNumber.

```
void  _XlcCopyFromArg( *src,  *dst,  size);
```

```
void  _XlcCopyToArg( *src,  **dst,  size);
```

Similar to `_XtCopyFromArg` and `_XtCopyToArg`.

```
void  _XlcCountVaList( var,  *count_ret);
```

Similar to `_XtCountVaList`.

```
void  _XlcVaToArgList( var,  count,  *args_ret);
```

Similar to `_XtVaToArgList`.

```
typedef struct _XlcResource {
      char *name;
      XrmQuark xrm_name;
      int size;
      int offset;
      unsigned long mask;
} XlcResource, *XlcResourceList;
```

```
#define    XlcCreateMask   (1L<<0)
#define    XlcDefaultMask  (1L<<1)
#define    XlcGetMask      (1L<<2)
#define    XlcSetMask      (1L<<3)
#define    XlcIgnoreMask   (1L<<4)
```

```
void  _XlcCompileResourceList( resources,  num_resources);
```

Similar to `_XtCompileResourceList`.

`char *` **`_XlcGetValues`**`(` *`base`*`,` *`resources`*`,` *`num_resources`*`,` *`args`*`,` *`num_args`*`,` *`mask`*`);`

Similar to XtGetSubvalues.

`char *` **`_XlcSetValues`**`(` *`base`*`,` *`resources`*`,` *`num_resources`*`,` *`args`*`,` *`num_args`*`,` *`mask`*`);`

Similar to XtSetSubvalues.

ANSI C Compatible Functions

The following are ANSI C/MSE Compatible Functions for non-ANSI C environment.

`int` **`_Xmblen`**`(` *`*str`*`,` *`len`*`);`

The `_Xmblen` function returns the number of characters pointed to by "*str*". Only "*len*" bytes in "*str*" are used in determining the character count returned. "*Str*" may point at characters from any valid codeset in the current locale.

The call `_Xmblen` is equivalent to _Xmbtowc(_Xmbtowc((*wchar_t**)NULL, *str*, *len*))

`int` **`_Xmbtowc`**`(` *`*wstr`*`,` *`*str`*`,` *`len`*`);`

The `_Xmbtowc` function converts the character(s) pointed to by "*str*" to their wide character representation(s) pointed to by "*wstr*". "*Len*" is the number of bytes in "*str*" to be converted. The return value is the number of characters converted.

The call `_Xmbtowc` is equivalent to _Xlcmbtowc((XLCd)NULL, *wstr*, *str*, *len*)

`int` **`_Xlcmbtowc`**`(` *`lcd`*`,` *`*wstr`*`,` *`*str`*`,` *`len`*`);`

The `_Xlcmbtowc` function is identical to `_Xmbtowc`, except that it requires the "*lcd*" argument. If "*lcd*" is (XLCd) NULL, `_Xlcmbtowc`, calls `_XlcCurrentLC` to determine the current locale.

`int` **`_Xwctomb`**`(` *`*str`*`,` *`wc`*`);`

The `_Xwctomb` function converts a single wide character pointed to by "*wc*" to its multibyte representation pointed to by "*str*". On success, the return value is 1.

The call `_Xwctomb` is equivalent to _Xlcwctomb((XLCd)NULL, *str*, *wstr*)

`int` **`_Xlcwctomb`**`(` *`lcd`*`,` *`*str`*`,` *`wc`*`);`

The `_Xlcwctomb` function is identical to _Xwctomb, except that it requires the "*lcd*" argument. If "*lcd*" is (XLCd) NULL, `_Xlcwctomb`, calls `_XlcCurrentLC` to determine the current locale.

`int` **`_Xmbstowcs`**`(` *`*wstr`*`,` *`*str`*`,` *`len`*`);`

The `_Xmbstowcs` function converts the NULL-terminated string pointed to by "*str*" to its wide character string representation pointed to by "*wstr*". "*Len*" is the number of characters in "*str*" to be converted.

The call `_Xmbstowcs` is equivalent to _Xlcmbstowcs((XLCd)NULL, *wstr*, *str*, *len*)

`int` **`_Xlcmbstowcs`**`(` *`lcd`*`,` *`*wstr`*`,` *`*str`*`,` *`len`*`);`

The `_Xlcmbstowcs` function is identical to _Xmbstowcs, except that it requires the "*lcd*" argument. If "*lcd*" is (XLCd) NULL, `_Xlcmbstowcs,` calls `_XlcCurrentLC` to determine the current locale.

```
int  _Xwcstombs( *str,  *wstr,  len);
```

The `_Xwcstombs` function converts the (wchar_t) NULL terminated wide character string pointed to by "*wstr*" to the NULL terminated multibyte string pointed to by "*str*".

The call `_Xwcstombs` is equivalent to _Xlcwcstombs((XLCd)NULL, *str, wstr, len*)

```
int  _Xlcwcstombs( lcd,  *str,  *wstr,  len);
```

The `_Xlcwcstombs` function is identical to _Xwcstombs, except that it requires the "*lcd*" argument. If "*lcd*" is (XLCd) NULL, `_Xlcwcstombs,` calls `_XlcCurrentLC` to determine the current locale.

```
int  _Xwcslen( *wstr);
```

The `_Xwcslen` function returns the count of wide characters in the (wchar_t) NULL terminated wide character string pointed to by "*wstr*".

```
wchar_t * _Xwcscpy( *wstr1,  *wstr2);
```

```
wchar_t *  _Xwcsncpy( *wstr1,  *wstr2,  len);
```

The `_Xwcscpy` function copies the (wchar_t) NULL terminated wide character string pointed to by "*wstr2*" to the object pointed at by "*wstr1*". "*Wstr1*" is (wchar_t) NULL terminated. The return value is a pointer to "*wstr1*".

The `_Xwcsncpy` function is identical to `_Xwcscpy,` except that it copies "*len*" wide characters from the object pointed to by "*wstr2*" to the object pointed to "*wstr1*".

```
int  _Xwcscmp( *wstr2);
```

```
int  _Xwcsncmp( *wstr2,  len);
```

The `_Xwcscmp` function compares two (wchar_t) NULL terminated wide character strings. The value returned is an integer less than, equal to, or greater than zero, depending on whether "*wstr1*" is lexicographicly less then, equal to, or greater than "*str2*".

The `_Xwcsncmp` function is identical to `_XlcCompareISOLatin1,` except that at most "*len*" wide characters are compared.