

locale

COLLABORATORS

	<i>TITLE :</i> locale		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		March 29, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	locale	1
1.1	locale.doc	1
1.2	locale.library/--environment_vars--	1
1.3	locale.library/--rexxhost--	2
1.4	locale.library/--structures--	2
1.5	locale.library/CloseCatalog	6
1.6	locale.library/CloseLocale	6
1.7	locale.library/ConvToLower	6
1.8	locale.library/ConvToUpper	7
1.9	locale.library/FormatDate	8
1.10	locale.library/FormatString	9
1.11	locale.library/GetCatalogStr	11
1.12	locale.library/GetLocaleStr	11
1.13	locale.library/IsXXXX	12
1.14	locale.library/OpenCatalog	13
1.15	locale.library/OpenLocale	15
1.16	locale.library/ParseDate	16
1.17	locale.library/StrConvert	17
1.18	locale.library/StrnCmp	17

Chapter 1

locale

1.1 locale.doc

```
--environment_vars-- ()
--rexxhost--
--structures--
CloseCatalog ()
CloseLocale ()
ConvToLower ()
ConvToUpper ()
FormatDate ()
FormatString ()
GetCatalogStr ()
GetLocaleStr ()
IsXXXX ()
OpenCatalog ()
OpenLocale ()
ParseDate ()
StrConvert ()
StrnCmp ()
```

1.2 locale.library/--environment_vars--

Starting with V40, locale.library maintains a global environment variable called "Language" which contains the name of the current default language as used in the system. This is the name of the language associated with the Locale structure returned by `OpenLocale(NULL)`.

EXAMPLE
From a shell:

```
Echo "The system language currently is: $Language"
```

will print the name of the current system language ("english", "français", etc)

1.3 locale.library/--rexxhost--

HOST INTERFACE

locale.library provides an ARexx function host interface that enables ARexx programs to take advantage of system localization. The functions provided by the interface are directly related to the functions described herein, with the differences mostly being in the way they are called.

The function host library vector is located at offset -30 from the library. This is the value you provide to ARexx in the AddLib() function call.

FUNCTIONS

```
CloseCatalog (CATALOG/N/A)
ConvToLower (CHARACTER/A)
ConvToUpper (CHARACTER/A)
GetCatalogStr (CATALOG/A, STRING/N/A, DEFAULT/A)
IsAlNum (CHARACTER/A)
IsAlpha (CHARACTER/A)
IsCntrl (CHARACTER/A)
IsDigit (CHARACTER/A)
IsGraph (CHARACTER/A)
IsLower (CHARACTER/A)
IsPrint (CHARACTER/A)
IsPunct (CHARACTER/A)
IsSpace (CHARACTER/A)
IsUpper (CHARACTER/A)
IsXDigit (CHARACTER/A)
OpenCatalog (NAME/A, BUILTINLANGUAGE/A, VERSION/N/A)
Strncmp (STRING1/A, STRING2/A, TYPE/N/A)
```

EXAMPLE

```
/* localetest.rexx */

/* Make sure locale is loaded as a function host */
IF ~SHOW(L,'locale.library') THEN DO
  CALL ADDLIB('locale.library',0,-30)
END;

say ConvToLower("A");
say ConvToUpper("b");
say IsAlpha("1");

catalog = OpenCatalog("sys/workbench.catalog","english",0);
say GetCatalogStr(catalog,34,"default");
say CloseCatalog(catalog);
say StrnCmp("test","test",2);
```

1.4 locale.library/--structures--

The Locale structure is the main public structure provided by locale.library. The structure is defined in <libraries/locale.h> and consists of the following fields:

STRPTR loc_LocaleName
Locale's name.

STRPTR loc_LanguageName
The language of the driver bound to this locale.

STRPTR loc_PrefLanguages[10]
The ordered list of preferred languages for this locale.

ULONG loc_Flags
Locale flags. Currently always 0.

ULONG loc_CodeSet
Specifies the code set required by this locale. Currently, this value is always 0.

ULONG loc_CountryCode
The international country code.

ULONG loc_TelephoneCode
The international telephone code for the country.

LONG loc_GMTOffset
The offset in minutes of the current location from GMT.
Positive indicates a Westerly direction from GMT,
negative Easterly.

UBYTE loc_MeasuringSystem
The measuring system being used.

STRPTR loc_DateTimeFormat
The date and time format string, ready to pass to FormatDate()

STRPTR loc_DateFormat
The date format string.

STRPTR loc_TimeFormat
The time format string.

STRPTR loc_ShortDateTimeFormat
The short date and time format string, ready to pass to FormatDate()

STRPTR loc_ShortDateFormat
The short date format string.

STRPTR loc_ShortTimeFormat
The short time format string.

STRPTR loc_DecimalPoint
The decimal point character used to format non-monetary quantities.

STRPTR loc_GroupSeparator
The characters used to separate groups of digits before the decimal-point character in formatted non-monetary quantities.

STRPTR loc_FracGroupSeparator

The characters used to separate groups of digits after the decimal-point character in formatted non-monetary quantities.

STRPTR loc_Grouping

A string whose elements indicate the size of each group of digits before the decimal-point character in formatted non-monetary quantities.

STRPTR loc_FracGrouping

A string whose elements indicate the size of each group of digits after the decimal-point character in formatted non-monetary quantities.

STRPTR loc_MonDecimalPoint

The decimal-point used to format monetary quantities.

STRPTR loc_MonGroupSeparator

The separator for groups of digits before the decimal-point in monetary quantities.

STRPTR loc_MonFracGroupSeparator

The separator for groups of digits after the decimal-point in monetary quantities.

STRPTR loc_MonGrouping

A string whose elements indicate the size of each group of digits before the decimal-point character in monetary quantities.

STRPTR loc_MonFracGrouping

A string whose elements indicate the size of each group of digits after the decimal-point character in monetary quantities.

UBYTE loc_MonFracDigits

The number of fractional digits (those after the decimal-point) to be displayed in a formatted monetary quantity.

UBYTE loc_MonIntFracDigits

The number of fractional digits (those after the decimal-point) to be displayed in an internationally formatted monetary quantity.

STRPTR loc_MonCS

The local currency symbol applicable to the current locale.

STRPTR loc_MonSmallCS

The currency symbol for small amounts.

STRPTR loc_MonIntCS

The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in ISO 4217 Codes for the Representation of Currency and Funds. The fourth character (immediately preceding the NULL) is the character used to separate the international currency symbol from the monetary quantity.

STRPTR loc_MonPositiveSign

The string used to indicate a non-negative monetary quantity.

UBYTE loc_MonPositiveSpaceSep

Specifies the number of spaces separating the currency symbol from the non-negative monetary quantity.

UBYTE loc_MonPositiveSignPos

Set to a value indicating the positioning of loc_MonPositiveSign for a non-negative monetary quantity.

UBYTE loc_MonPositiveCSPos

Set to 1 or 0 if loc_MonCS respectively precedes or succeeds the value for a non-negative monetary quantity.

STRPTR loc_MonNegativeSign

The string used to indicate a negative monetary quantity.

UBYTE loc_MonNegativeSpaceSep

Specifies the number of spaces separating the currency symbol from the negative monetary quantity.

UBYTE loc_MonNegativeSignPos

Set to a value indicating the positioning of loc_MonNegativeSign for a negative monetary quantity.

UBYTE loc_MonNegativeCSPos

Set to 1 or 0 if loc_MonCS respectively precedes or succeeds the value for a negative monetary quantity.

The grouping tables pointed to by loc_Grouping, loc_FracGrouping, loc_MonGrouping, and loc_MonFracGrouping contain a stream of bytes with the following values:

255 No further grouping is to be performed.

0 The previous element is to be repeatedly used for the remainder of the digits.

1..254 The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The values of loc_MonPositiveSignPos and loc_MonNegativeSignPos are interpreted according to the following:

0 Parentheses surround the quantity and currency symbol

1 The sign string precedes the quantity and currency symbol

2 The sign string succeeds the quantity and currency symbol

3 The sign string immediately precedes the currency symbol

4 The sign string immediately succeeds the currency symbol.

1.5 locale.library/CloseCatalog

NAME

CloseCatalog -- close a message catalog. (V38)

SYNOPSIS

```
CloseCatalog(catalog);  
            A0
```

```
VOID CloseCatalog(struct Catalog *);
```

FUNCTION

Concludes access to a message catalog. The usage count of the catalog is decremented. When this count reaches 0, the catalog can be expunged from system memory whenever a memory panic occurs.

INPUTS

catalog - the message catalog to close. A NULL catalog is a valid parameter and is simply ignored.

SEE ALSO

OpenCatalog(), GetCatalogStr()

1.6 locale.library/CloseLocale

NAME

CloseLocale -- close a locale. (V38)

SYNOPSIS

```
CloseLocale(locale);  
            A0
```

```
VOID CloseLocale(struct Locale *);
```

FUNCTION

Concludes access to a locale.

INPUTS

locale - an opened locale. A NULL locale is a valid parameter and is simply ignored.

SEE ALSO

OpenLocale(), <libraries/locale.h>

1.7 locale.library/ConvToLower

NAME

ConvToLower -- convert a character to lower case. (V38)

SYNOPSIS

```
char = ConvToLower(locale,character);
```

D0 A0 D0

```
ULONG ConvToLower(struct Locale *,ULONG);
```

FUNCTION

This function tests if the character specified is upper case. If it is then the lower case version of that character is returned, and if it isn't then the original character is returned.

INPUTS

locale - the locale to use for the conversion

character - the character to convert

RESULT

char - a (possibly) converted character

NOTE

This function requires a full 32-bit character be passed-in in order to support multi-byte character sets.

1.8 locale.library/ConvToUpper

NAME

ConvToUpper -- convert a character to upper case. (V38)

SYNOPSIS

```
char = ConvToUpper(locale,character);
```

D0 A0 D0

```
ULONG ConvToUpper(struct Locale *,ULONG);
```

FUNCTION

This function tests if the character specified is lower case. If it is then the upper case version of that character is returned, and if it isn't then the original character is returned.

INPUTS

locale - the locale to use for the conversion

character - the character to convert

RESULT

char - a (possibly) converted character

NOTE

This function requires a full 32-bit character be passed-in in order to support multi-byte character sets.

1.9 locale.library/FormatDate

NAME

FormatDate -- generate a date string based on a date formatting template. (V38)

SYNOPSIS

```
FormatDate(locale, fmtTemplate, date, putCharFunc);
           A0      A1          A2    A3
```

```
VOID FormatDate(struct Locale *, STRPTR, struct DateStamp *,
               struct Hook *);
```

FUNCTION

This function processes a formatting template and generates a stream of bytes that's sent one character at a time to the putCharFunc callback hook.

INPUTS

locale - the locale to use for the formatting
 fmtTemplate - the NULL-terminated template describing the desired format for the date. This is constructed just like C-language printf() statements, except that different formatting codes are used. Just like in C, formatting codes start with a % followed by the formatting command. The following commands are accepted by this function:

```
%a - abbreviated weekday name
%A - weekday name
%b - abbreviated month name
%B - month name
%c - same as "%a %b %d %H:%M:%S %Y"
%C - same as "%a %b %e %T %Z %Y"
%d - day number with leading 0s
%D - same as "%m/%d/%y"
%e - day number with leading spaces
%h - abbreviated month name
%H - hour using 24-hour style with leading 0s
%I - hour using 12-hour style with leading 0s
%j - julian date
%m - month number with leading 0s
%M - the number of minutes with leading 0s
%n - insert a linefeed
%p - AM or PM strings
%q - hour using 24-hour style
%Q - hour using 12-hour style
%r - same as "%I:%M:%S %p"
%R - same as "%H:%M"
%S - number of seconds with leadings 0s
%t - insert a tab character
%T - same as "%H:%M:%S"
%U - week number, taking Sunday as first day of week
%w - weekday number
%W - week number, taking Monday as first day of week
%x - same as "%m/%d/%y"
%X - same as "%H:%M:%S"
```

%y - year using two digits with leading 0s
 %Y - year using four digits with leading 0s

If the template parameter is NULL, a single NULL byte is sent to putCharFunc.
 date - the date to format into a string
 putCharFunc - a callback hook invoked for every character generated, including for the terminating NULL character. The hook is called with:

A0 - address of Hook structure
 A1 - character for hook to process (not a pointer!)
 A2 - locale pointer

SEE ALSO
 ParseDate(), <libraries/locale.h>, <dos/dos.h>

1.10 locale.library/FormatString

NAME

FormatString -- format data into a character stream. (V38)

SYNOPSIS

```
next = FormatString(locale,fmtTemplate,dataStream,putCharFunc);
```

D0 A0 A1 A2 A3

```
APTR FormatString(struct Locale *,STRPTR,APTR,struct Hook *);
```

FUNCTION

This function performs C-language-like formatting of a data stream, outputting the result a character at a time. Where % formatting commands are found in the formatting template, they are replaced with the corresponding elements in 'dataStream'. %% must be used in the string if a % is desired in the output.

An extension to the standard C-language printf() conventions used by FormatString() is argument position specification. Specifying the argument position lets the order of the % commands change while the arguments provided remain the same. Using the C printf() call as an example:

```
printf("%d eyes, %d feet and %d ears",eyes,feet,ears);  
printf("%3$d ears, %1$d eyes and %2$d feet",eyes,feet,ears);
```

These two statements would produce the following output:

```
"2 eyes, 3 feet and 4 ears" for the first  
"4 ears, 2 eyes and 3 feet" for the second
```

The argument positioning feature lets you change the format string being processed while keeping the data stream the same. This is an invaluable tool when translating strings to different languages.

INPUTS

locale - the locale to use for the formatting
 fmtTemplate - a C-language-like NULL-terminated format string, with the following supported % options:

`%[arg_pos$][flags][width][.limit][length]type`

`arg_pos` - ordinal position of the argument for this command within the array of arguments pointed to by 'dataStream'

`$` - must follow the `arg_pos` value, if specified

`flags` - only one allowed. '-' specifies left justification.

`width` - field width. If the first character is a '0', the field is padded with leading 0s.

`.` - must precede the field limit value, if specified

`limit` - maximum number of characters to output from a string. (only valid for %s or %b).

`length` - size of input data defaults to word (16-bit) for types c, d, u and x, 'l' changes this to long (32-bit).

`type` - supported types are:

- b - BSTR, data is 32-bit BPTR to byte count followed by a byte string. A NULL BPTR is treated as an empty string.
- d - signed decimal
- D - signed decimal using the locale's formatting conventions
- u - unsigned decimal
- U - unsigned decimal using the locale's formatting conventions
- x - hexadecimal with hex digits in uppercase
- X - hexadecimal with hex digits in lowercase
- s - string, a 32-bit pointer to a NULL-terminated byte string. A NULL pointer is treated as an empty string.
- c - character

If the formatting template parameter is NULL, the function returns without outputting anything. Note the meaning of %x and %X are swapped with respect to standard C conventions. This is for compatibility with `exec.library/RawDoFmt()`.

`dataStream` - a stream of data that is interpreted according to the format string. Often this is a pointer into the task's stack.

`putCharFunc` - a callback hook invoked for every character generated, including for the terminating NULL character. The hook is called with:

- A0 - address of Hook structure
- A1 - character for hook to process (not a pointer!)
- A2 - locale pointer

the function is called with a NULL char at the end of the format string.

RESULT

`next` - A pointer to beyond the last data element used in 'dataStream' (the next argument that would have been processed). This allows multiple formatting passes to be made using the same data.

WARNING

This function formats word values in the data stream. If your compiler defaults to longs, you must add an "l" to your specifications. This can get strange for characters, which might look like "%lc".

SEE ALSO
exec.library/RawDoFmt()

1.11 locale.library/GetCatalogStr

NAME
GetCatalogStr -- get a string from a message catalog. (V38)

SYNOPSIS
string = GetCatalogStr(catalog,stringNum,defaultString);
D0 A0 D0 A1

STRPTR GetCatalogStr(struct Catalog *,LONG,STRPTR);

FUNCTION
This function returns a specific string within a message catalog. If the catalog parameter is NULL, or the requested message does not exist, then defaultString is returned.

INPUTS
catalog - a message catalog as obtained from OpenCatalog(), or NULL
stringNum - a message number within the catalog
defaultString - string to return in case "catalog" is NULL or "stringNum" can't be found

RESULT
string - a pointer to a NULL-terminated string. The returned string is READ-ONLY, do NOT modify! This string pointer is valid only as long as the catalog remains open.

SEE ALSO
OpenCatalog(), CloseCatalog()

1.12 locale.library/GetLocaleStr

NAME
GetLocaleStr -- get a standard string from a locale. (V38)

SYNOPSIS
string = GetLocaleStr(locale,stringNum);
D0 A0 D0

STRPTR GetLocaleStr(struct Locale *,ULONG);

FUNCTION
This function returns a specific string associated with the given locale.

INPUTS

locale - a valid locale
stringNum - the number of the string to get a pointer to. See the constants defined in <libraries/locale.h> for the possible values.

RESULT

string - a pointer to a NULL-terminated string, or NULL if the requested string number was out of bounds. The returned string is READ-ONLY, do NOT modify! This string pointer is valid only as long as the locale remains open.

SEE ALSO

OpenLocale(), CloseLocale(), <libraries/locale.h>

1.13 locale.library/IsXXXX

NAME

IsXXXX -- determine whether a character is of a certain type. (V38)

SYNOPSIS

```
state = IsXXXX(locale,character);
```

```
D0          A0          D0
```

```
BOOL IsXXXX(struct Locale *,ULONG);
```

FUNCTION

These functions determine whether the character specified is of a certain type, according to the supplied locale.

IsAlNum() - test if alphanumeric character
IsAlpha() - test if alphabetical character
IsCntrl() - test if control character
IsDigit() - test if decimal digit character
IsGraph() - test if visible character
IsLower() - test if lower case character
IsPrint() - test if blank
IsPunct() - test if punctuation character
IsSpace() - test if white space character
IsUpper() - test if upper case character
IsXDigit() - test if hexadecimal digit

INPUTS

locale - the locale to use for the test
character - the character to test

RESULT

state - TRUE if the character is of the required type, FALSE otherwise

NOTE

These functions require full 32-bit characters be passed-in in order to support multi-byte character sets.

1.14 locale.library/OpenCatalog

NAME

OpenCatalogA -- open a message catalog. (V38)
 OpenCatalog -- varargs stub for OpenCatalogA(). (V38)

SYNOPSIS

```
catalog = OpenCatalogA(locale,name,tagList);
D0                      A0      A1      A2

struct Catalog *OpenCatalogA(struct Locale *,STRPTR,struct TagItem *);

catalog = OpenCatalog(locale,name,firstTag, ...);

struct Catalog *OpenCatalog(struct Locale *,STRPTR,Tag, ...);
```

FUNCTION

This function opens a message catalog. Catalogs contain all the text strings that an application uses. These strings can easily be replaced by strings in a different language, which causes the application to magically start operating in that new language.

Catalogs originally come from disk files. This function searches for them in the following places:

```
PROGDIR:Catalogs/languageName/name
LOCALE:Catalogs/languageName/name
```

where languageName is the name of the language associated with the locale parameter. So assuming an application called WizPaint:

```
catalog = OpenCatalog(NULL,
    "WizPaint.catalog",
    OC_BuiltInLanguage,"english",
    TAG_DONE);
```

Passing NULL as first parameter to OpenCatalog() indicates you wish to use the system's default locale. Assuming the default locale specifies "deutsch" as language, OpenCatalog() tries to open the catalog as:

```
PROGDIR:Catalogs/deutsch/WizPaint.catalog
```

and if that file is not found, then OpenCatalog() tries to open it as:

```
LOCALE:Catalogs/deutsch/WizPaint.catalog
```

PROGDIR: is not always checked before LOCALE: is. If the volume which PROGDIR: is assigned to is NOT currently mounted, and if the one which LOCALE: is assigned to IS mounted, then LOCALE: is checked first, followed by PROGDIR: if needed. This is done in order to minimize the number of disk swaps on floppy systems.

The OC_BuiltInLanguage tag specifies the language of the strings that are built into the application. If the language of the

built-in strings matches that of the locale, then no catalog need be loaded from disk and the built-in strings can be used directly.

locale.library caches text catalogs in order to minimize disk access. As such, OpenCatalog() may or may not cause disk access. This fact should be taken into consideration. Unused catalogs are automatically flushed from the system when there is not enough memory. When there is disk access, it is possible a DOS requester may be opened asking for a volume to be inserted. You can avoid this requester opening by setting your process' pr_WindowPtr field to -1.

INPUTS

locale - the locale for which the catalog should be opened, or NULL. When NULL, then the system's default locale is used. This should generally be NULL
name - the NULL-terminated name of the catalog to open, typically the application name with a ".catalog" extension
tagList - pointer to an array of tags providing optional extra parameters, or NULL

TAGS

OC_BuiltInLanguage (STRPTR) - language of built-in strings of the application. That is, this tag identifies the language used for the "defaultString" parameter used in the GetCatalogStr() function. Default is "english". Providing this tag and setting its value to NULL indicates that there are no built-in strings.

OC_BuiltInCodeSet (ULONG) - code set of built-in strings. Default is 0. THIS TAG SHOULD ALWAYS BE SET TO 0 FOR NOW.

OC_Language (STRPTR) - language explicitly requested for the catalog. A catalog of this language will be returned if possible, otherwise a catalog in one of the user's preferred languages. This tag should normally not be provided as it overrides the user's preferences.

OC_Version (UWORD) - catalog version number required. Default is 0 which means to accept any version of the catalog that is found. Note that if a version is specified, the catalog's version must match it exactly. This is different from version numbers used by OpenLibrary().

RESULT

catalog - a message catalog to use with GetCatalogStr() or NULL. A NULL result does not necessarily indicate an error. If OpenCatalog() determines that the built-in strings of the application can be used instead of an external catalog from disk, then NULL is returned. To determine whether a NULL result actually indicates an error, look at the return value of dos.library/IOErr(). 0 means no error.

GetCatalogStr() interprets a NULL catalog as meaning to use the built-in strings.

NOTE

In most cases, failing to open a catalog should not be considered a fatal error, and the application should continue operating and simply use the built-in set of strings instead of the disk-based catalog. Note that GetCatalogStr() accepts a NULL catalog pointer for this very reason.

Also note that displaying an error message when a catalog fails to open can be a meaningless endeavor as the message is likely in a language the user does not understand.

SEE ALSO

CloseCatalog(), GetCatalogStr()

1.15 locale.library/OpenLocale

NAME

OpenLocale -- open a locale. (V38)

SYNOPSIS

```
locale = OpenLocale(name);
```

```
D0                                A0
```

```
struct Locale *OpenLocale(STRPTR);
```

FUNCTION

This function opens a named locale. Locales contain many parameters that an application needs to consider when being integrated into different languages, territories and customs. Using the information stored in a locale instead of hard-coding it into an application, lets the application dynamically adapt to the user's environment.

Locales originally come from disk files which are created by the user using the Locale preferences editor. Passing a NULL instead of a name causes this function to return the current default locale. This is what most applications will do.

Every locale specifies a language, and special language drivers must be loaded from disk depending on which language is being used. These files include for example:

```
LOCALE:Languages/français.language
LOCALE:Languages/dansk.language
LOCALE:Languages/italiano.language
```

INPUTS

name - the NULL-terminated name of the locale to open, or NULL to open the current default locale. This should generally be NULL. The name you supply must be a pathname leading to a locale preferences file. This is an IFF PREF file as saved by Locale prefs, that can contain both LCLE and CTRY chunks. See <prefs/locale.h> for definitions.

RESULT

locale - a pointer to an initialized Locale structure, or NULL if the locale could not be loaded. In the case of a NULL return, the DOS IoErr() function can be called to obtain more information on the failure.

When passing a NULL name parameter to this function, you are guaranteed a valid return.

SEE ALSO

CloseLocale(), <libraries/locale.h>, <prefs/locale.h>

1.16 locale.library/ParseDate

NAME

ParseDate -- interpret a string according to the date formatting template and convert it into a DateStamp. (V38)

SYNOPSIS

```
state = ParseDate(locale, date, fmtTemplate, getCharFunc);
```

```
D0          A0      A1      A2          A3
```

```
BOOL ParseDate(struct Locale *, struct DateStamp *, STRPTR, struct Hook *);
```

FUNCTION

This function converts a stream of characters into an AmigaDOS DateStamp structure. The characters are obtained from the getCharFunc callback hook and the formatting template is used to direct the parse.

INPUTS

locale - the locale to use for the formatting

date - place to put the converted date, this may be NULL in which case this routine can be used to simply validate a date

fmtTemplate - the date template describing the expected format of the data. See FormatDate() documentation for a description of date templates. The following formatting controls from FormatDate() can be used in ParseDate():

```
%a %A %b %B %d %e %h %H %I %m %M %p %S %y %Y
```

getCharFunc - a callback hook invoked whenever a character is required.

The hook should return the next character to process, with a NULL character to indicate the end of the string. The hook is called with:

A0 - address of Hook structure

A1 - locale pointer

A2 - NULL

The hook returns the character to process in D0. Note that a complete 32-bit result is expected in D0, not just 8 bits.

RESULT

state - TRUE if the parsing went OK, or FALSE if the input did not

match the template

SEE ALSO

FormatDate(), <dos/dos.h>

1.17 locale.library/StrConvert

NAME

StrConvert -- transform a string according to collation information.
(V38)

SYNOPSIS

```
length = StrConvert(locale,string,buffer,bufferSize,type);
D0              A0      A1      A2      D0      D1
```

```
ULONG StrConvert(struct Locale *,STRPTR,APTR,ULONG,ULONG);
```

FUNCTION

This function transforms the passed string and places the resulting into the supplied buffer. No more than bufferSize bytes are copied into the buffer.

The transformation is such that if the C strcmp() function is applied to two transformed strings, it returns a value corresponding to the result returned by the StrnCmp() function applied to the two original strings.

INPUTS

locale - the locale to use for the transformation
string - NULL-terminated string to transform
buffer - buffer where to put the transformed string
bufferSize - maximum number of bytes to deposit in the buffer

StrConvert() may require more storage than
the unconverted string does

type - describes how the transformation is to be performed. See
the documentation on StrnCmp() for more information on the
comparison types available

RESULT

length - length of the transformed string which is the number of bytes
deposited in the buffer minus 1 (since strings are NULL-
terminated)

SEE ALSO

StrnCmp(), <libraries/locale.h>

1.18 locale.library/StrnCmp

NAME

StrnCmp -- localized string comparison. (V38)

SYNOPSIS

```
result = StrnCmp(locale, string1, string2, length, type);
D0          A0      A1      A2      D0      D1
```

```
LONG StrnCmp(struct Locale *, STRPTR, STRPTR, LONG, ULONG);
```

FUNCTION

Compares string1 to string2 according to the collation information provided by the locale and returns an integer greater than, equal to, or less than zero, accordingly as the string pointed to by string1 is greater than, equal to, or less than the string pointed to by string2.

The length parameter specifies how many characters to compare, or if the length is specified as -1 then the strings are compared until a NULL is encountered.

The type parameter dictates how the comparison is to be performed.

INPUTS

locale - the locale to use for this comparison
string1 - NULL-terminated string
string2 - NULL-terminated string
length - the maximum number of characters to be compared, or -1 to compare all characters until a NULL is encountered
type - describes how the comparison is to be performed. The following values can be passed:
SC_ASCII causes an ASCII-based case-insensitive comparison to be performed. SC_ASCII is the fastest of the comparison types, but it uses ASCII ordering and considers accented characters different than their non-accented counterparts.

SC_COLLATE1 causes the characters to be compared using their primary sorting order. This effectively produces a comparison that ignores letter case and diacritical marks. That is, letters such as "e" and "é" are treated as if they were both "e".

SC_COLLATE2 causes the characters to be compared using both their primary and secondary sorting order. SC_COLLATE2 is slower than SC_COLLATE1. This is the type of comparison to use when sorting data to be presented to the user. It operates in two passes. First it performs a comparison equivalent to SC_COLLATE1. If both strings compare the same, then a second pass is made using the secondary sorting order, which gives finer resolution to the comparison. For example, SC_COLLATE1 would return the following strings as identical:

"père" and "pere"

since SC_COLLATE1 ignores diacritical marks. SC_COLLATE2 would make a second pass over the string comparing diacritical marks instead of actual characters.

RESULT

result - relationship between string1 and string2
<0 means string1 < string2
=0 means string1 = string2
>0 means string1 > string2

SEE ALSO

`OpenLocale()`, `CloseLocale()`, `StrConvert()`