Internationalization Service Group

Internationalization: Overview

Internationalization provides the functionality for adapting programs to various languages and localities and for making conversions to use the internationally recognized Unicode* standard.

Internationalization

Internationalization provides functionality to examine and manipulate strings in various language formats. Functions are provided to support both single and multi-byte characters, as well as to support various locale conventions, such as decimal separations and time format. To access information about internationalization, begin with Internationalization: Guides.

Unicode

The Unicode API is divided into two components:

Unicode Table Group

Available for some time, this component is built around text files called Unicode tables that provide access to Unicode/code page conversions. The function set is defined in nwunicode.h.

Unicode Converter Group

Recently released, this component is built around converters implemented as DLLs. With the "extended" functions, the component provides significantly increased control over the conversion processes. This function set is defined in nwnunicode.h.

Both components are operational, and both are fully supported. In addition, both header files provide access to Unicode string utility functions, such as **unicat**, **unichr**, and **unirev**. However, the Unicode Converter API offers important advantages. We suggest that you begin using it for new development.

NOTE: Although both sets can be used in the same application, they cannot both be used in the same file.

You can access information through the following guides:

Unicode Table API: Guides

Unicode Converter API: Guides

Internationalization

Internationalization: Guides

Internationalization: Concepts Guide

Internationalization: Tasks Guide

Types of Internationalization Functions

The Internationalization functions are divided into the following categories:

ANSI Compatibility Functions

Locale Information Functions

Internationalized String Functions

Internationalized Collation Functions

Internationalized Money Functions

Internationalized printf Functions

Internationalized ctype.h Functions

Multibyte and Double-Byte Functions

NLM Specific Internationalization Functions

Internationalization: Tasks

Setting up for International String Manipulation

Setting up for international string manipulation involve the following steps:

1. **Call NWLsetlocale.**

   This function initializes internal data structures according to the currently implementation-defined native locale. It returns the country code of the native locale for which the server is currently set.

   While C programs use the "C" locale by default, an application can designate the machine's native locale with **NWLsetlocale**. A number of standard functions are sensitive to the current locale setting. For example, when the locale is changed, the functions in ctype.h change how they check characters.

   > **CAUTION: Because NWLsetlocale initializes the structures containing the locale information, do not call any other Internationalization function until you have called NWLsetlocale .**

2. **Call NWLlocaleconv**

   If the Internationalization functions themselves do not provide all the locale-sensitive operations needed, you can call **NWLlocaleconv** to get raw locale data to write your own locale-sensitive functions.

   In addition, if you do not like the format for the locale as it is currently arranged, you can change it with **NWLlocaleconv**.

3. **Replace existing functions with Internationalization functions**

   See "ANSI Compatibility Functions" for the NetWare functions you should replace the ANSI-based functions with.

Parent Topic:

Internationalization: Guides

Internationalization: Concepts

ANSI Compatibility Functions

The following table lists all the ANSI functions that are used for internationalizing NetWare applications. A few of these have been adapted to special Novell® conditions, but most of them have remained the same.

Table auto. ANSI-based Internationalization Functions

NetWare Function	ANSI-based Function
NWLlocaleconv	**localeconv**
NWLmblen	**mblen**
NWLsetlocale	**setlocale**
NWLstrchr	**strchr**
NWLstrcoll	**strcoll**
NWLstrcspn	**strcspn**
NWLstrftime	**strftime**
NWLstrpbrk	**strpbrk**
NWLstrrchr	**strrchr**
NWLstrrev	**strrev**
NWLstrspn	**strspn**
NWLstrstr	**strstr**
NWLstrupr	**strupr**
NWLstrxfrm	**strxfrm**

An example of an ANSI function that has been adapted for NetWare is **NWLlocaleconv**. The amount of information it provides has been increased. Also, the structure element data types have been changed from pointers to character arrays; in enhanced mode, Windows cannot tolerate them as pointers.

For ANSI descriptions of these ANSI-based internationalization functions, refer to the specification *ANSI X3.159-1989*.

Parent Topic:

Types of Internationalization Functions

Functions that Support Parameter Reordering

The following NetWare internationalization functions, defined in nwlocale.h, support parameter reordering:

NWfprintf

NWprintf

NWsprintf

NWvfprintf

NWvprintf

NWvsprintf

In addition, the following functions, defined in nwsnut.h support parameter reordering in for the NetWare NLM user interface:

NWSAlert

NWSAlertWithHelp

NWSDisplayErrorText

NWSDisplayErrorCondition

NWSTrace

Internationalized Collation Functions

These functions perform collation operations and are affected by the LC_COLLATE locale data. Use them in place of their counterparts in string.h.

Function	Comment
NWGetCollateTable	Gets the Character Collation Table from the current operating system.
NWLstrcoll	Does a locale-sensitive string comparison of two strings.
NWstrncoll	Returns the difference in weight value between a string for which the collation value is known and a string for which it's not known.
NWLstrxfrm	Transforms a string by replacing each character with its corresponding collation value.

Parent Topic:

Types of Internationalization Functions

Internationalized ctype.h Functions

These functions perform alphanumeric tests and conversions. Use them in place of their counterparts in ctype.h.

Function	Comment
NWatoi (client only) **NWLatoi** (NLM only)	Converts a string of digits into an integer value. (Double-byte characters aren't converted.)
NWisalnum (client only) **NWLisalnum** (NLM only)	Returns whether a given value is alphabetic (A to Z or a to z) or a digit (0 to 9).
NWisalpha (client only) **NWLisalpha** (NLM only)	Returns whether a given value is alphabetic (A to Z or a to z).
NWisdigit (client only) **NWLisdigit** (NLM only)	Returns whether a given value is a digit (0 to 9).
NWIsxdigit (client only)	Returns whether a given value is a hexadecimal digit.

Parent Topic:

Internationalization: Guides

Internationalized Money Functions

These functions return monetary formats. Monetary formats are determined by the LC_MONETARY category of locale data.

Function	Comment
NWstrImoney	Gets the country prefix and money format for a numerical value.
NWstrmoney	Gets the locale-sensitive money format for a numerical value.

Parent Topic:

Types of Internationalization Functions

Internationalized printf Functions

These functions perform **printf**-style operations. They provide for the reordering of parameters on the stack before outputting data. Use them in place of their counterparts in stdio.h.

NetWare Function	ANSI printf Function
NWprintf	**printf**
NWfprintf	**fprintf**
NWsprintf	**sprintf**
NWvprintf	**vprintf**
NWvfprintf	**vfprintf**
NWvsprintf	**vsprintf**

Parent Topic:

Types of Internationalization Functions

Internationalized String Functions

These functions perform string operations. Use them in place of their counterparts in string.h.

Function	Comment
NWLstrbcpy	Copies a locale-sensitive string for a specified number of bytes (not characters).
NWLstrchr	Finds a character in a string.
NWLstrcspn	Computes the length of the maximum initial segment of one string consisting entirely of the characters not from another string.
NWLstrftime	Formats time and date according to the specified format.
NWLstricmp (client)	Performs a case-sensitive comparison of two strings.
NWstrlen (client)	Returns the number of bytes in a string.
NWLstrlwr (client)	Converts a string to lower case using locale

	information.
NWstrncpy	Copies a locale-sensitive string for a specified number of characters (not bytes).
NWstrnum	Formats a number for a specific country and returns the number in a string.
NWLstrpbrk	Locates the first occurrence in a string of any character from another string.
NWLstrrchr	Locates the last occurrence of a character in a string.
NWLstrrev	Reverses the order of the characters in a string.
NWLstrspn	Computes the length of the maximum initial segment of one string consisting entirely of characters from another string.
NWLstrstr	Searches one string for another string.
NWLstrtok (client)	Finds the next token in a string.
NWLstrupr	Converts a string to upper case using locale information.
NWLTruncateString (client)	Truncates a string at the specified number of characters.
NWultoa (client)	Converts a long unsigned integer to a string.
NWutoa (client)	Converts an unsigned integer to a string.
NWitoa (client)	Converts an integer to a string.
NWltoa (client)	Converts a long integer to a string.
NWLmbslen (client)	Counts the number of characters (not bytes) in a string.

Parent Topic:

Types of Internationalization Functions

Internationalization Include Files

Include nwlocale.h in any application that calls a Novell internationalization function.

If your NetWare 3.x NLM uses register-based parameter-passing, you must include cdecl.h, to which the Novell internationalization functions have been added.

NOTE: For NetWare 4.x, the "NWL" functions are provided by clib.nlm. For NetWare 3.12 these functions are provided by after311.nlm . The "NWL" functions also work on NetWare 3.11 if after311.nlm is loaded, but the end user might not have after311.nlm, since after311.nlm does not ship with the 3.11 OS.

Parent Topic:

Internationalization: Guides

Internationalization Overview

Internationalization is the strategy for adapting programs to diverse character sets and localities. The standard C contribution to internationalization is the header file locale.h. This file defines the lconv structure that controls formatting of numeric values for things like time and monetary expressions.

You can use internationalization functions declared in nwlocale.h in place of the standard C services found in locale.h. By using nwlocale.h, you not only receive the adaptive numeric formatting associated with locale.h, but also gain support for double-byte encoding schemes used to support large Japanese character sets. Because of the pervasive effect of a locale on standard input and output, internationalization functions in nwlocale.h also replace the print and scan functions declared in stdio.h.

Parent Topic:

Internationalization: Guides

Locale

A **locale** is the area of the world in which an application user's native language is spoken. Most of the time, a locale is a certain country: United States, Denmark, South Korea, etc.

Parent Topic:

Internationalization: Guides

Related Topics:

NetWare Locale Information Additions

Locale-Sensitive Data

Locale Information Functions

These functions access LCONV to initialize the locale.

Function	**Comment**

NWGetNWLOCALEVersion (client only)	Returns the library version.
NWLlocaleconv	Sets the elements of an LCONV structure according to the current locale setting.
NWLsetlocale	Initializes the locale information.
NWLmblen	Counts the characters (not bytes) in a string.

Parent Topic:

Types of Internationalization Functions

Locale-Sensitive Data

Use Internationalization to enable categories of information within a locale. Conceivably, an application may need to internationalize some categories of information and not others. Each category can be set individually. When you specify a category of locale information, only the functions related to that category are affected. For example, if you set the locale for LC_CTYPE, only the functions defined by ctype.h are internationalized. The following categories are defined:

LC_ALL (client only)NLC_ALL (client and NLM)	Set all categories
LC_COLLATE (client only) NLC_COLATE (client and NLM)	Set only data related to collation functions
LC_CTYPE (client only) NLC_CTYPE (client and NLM)	Set only data related to ctype.h
LC_MONETARY (client only) NLC_MONETARY (client and NLM)	Set only data related to currency
LC_NUMERIC (client only) NLC_NUMERIC (client and NLM)	Set only data related to numeric format
LC_TIME (client only) NLC_TIME (client and NLM)	Set only data related to time format

The following table shows the various sequences for formatting information and the separators used to mark numeric divisions in three locales. Note that locales can share the same formatting conventions in some categories and not others.

Table auto. Locale-Sensitive Data

Locale	**Date**	**Time**	**Number**	**Currency**
Australia	23/8/94	15:12:28	1,234.22	$1.22
Denmark	23-08-94	15.12.28	1.234,22	1,22 kr
South Korea	94.8.23	3:12:28 PM	1,234.22	W1.22

Parent Topic:

Locale

Multibyte and Double-Byte Functions

These functions scan characters in multibyte strings.

Function	**Comment**
NWCharType	Determines whether a character is single- or double-byte.
NWCharLwr (client only)	Converts a character to lower case. The result is based on the country information table.
NWCharUpr	Converts a character to upper case. The result is based on the country information table.
NWCharVal	Gets the integer value of a character in a string. Handles double-byte characters correctly.
NWIncrement	Increments a multibyte string pointer by the specified number of characters.
NWLInsertChar (client only)	Inserts a character at a given position in a string.
NWNextChar	Increments a pointer to the next character in a multibyte string. (Implemented as a macro in DOS and OS/2.)
NWPrevChar	Finds the beginning of the nearest previous character in a multibyte string.

Related Topics:

Types of Internationalization Functions

Parsing a double-byte string: Example

Native Locales

The format in which the system presents certain common pieces of information---date, time, numbers, currency---varies from locale to locale. The following table illustrates how these vary.

Table auto. Sample Formats from Three Native Locales

Locale	Date	Time	Number	Currency
Australia	23/8/94	15:12:28	1,234.22	$1.22
Denmark	23-08-94	15.12.28	1.234,22	1,22 kr
South Korea	94.8.23	3:12:28 PM	1,234.22	W1.22

In Australia, the commonly used formats resemble those of the United States except for the time format. The time format is 24-hour and therefore does not need "AM" or "PM" in order to be understood. Notice that Denmark's time is also 24-hour, but South Korea's is 12-hour.

Notice that the date formats, including dividers, for the three countries are all different: Australia's is day/month/year, Denmark's is day-month-year, and South Korea's is year.month.day.

Number formats are important because the position of the comma and period frequently switch from locale to locale.

> **NOTE:** To check which native locale your server is set for, use **NWLsetlocale**.
>
> In a NetWare server environment, the current locale is determined at OS boot-time. The locale cannot be altered without rebooting the OS.

Parent Topic:

Internationalization: Guides

NetWare Locale Information Additions

In addition to the fields found in the lconv structure, the LCONV structure adds the following information.

Table auto. NetWare Locale Data

Field	**Type**	**Comment**
code_page	int	Code page ID value.
country_id	int	Country code ID value.
data_list_separator	char[2]	Character used to separate items in a list.
date_separator	char[2]	Character used to separate date values.
time_separator	char[2]	Character used to separate time values.
time_format	char	0 = 12 hour, 1 = 24 hour clock
date_format	int	0 = MDY, 1 = DMY, 2 = YMD
reserved	char[40]	Undefined.

Parent Topic:

Locale

NLM Specific Functions

The functions in the following table are specific to NLM applications.

Table auto. NLM Specific Internationalization Functions

Function	**Meaning**
AddLanguage	Adds a language to the OS supported language list See Adding to OS Supported Language List: Example.
GetCurrentOSLanguageID	Returns the language ID for the language currently running on the server
LoadLanguageMessageTable	Loads a pointer to a language message table so that the table can be used for language enabling support
RenameLanguage	Changes the name string associated with an OS language ID
ReturnLanguageName	Returns the name associated with an OS language ID
SetCurrentOSLanguageID	Sets the language ID for the language currently running on the server

Parent Topic:

Types of Internationalization Functions

Internationalization: Functions

AddLanguage

Adds a language to the OS supported language list
Local Servers: nonblocking
Remote Servers: N/A
Classification: 4.x
Platform: NLM
SMP Aware: No
Service: Internationalization

Syntax

```
#include <nwadv.h>

int AddLanguage (
   int    languageID,
   BYTE  *newLanguageName,
   int    showErrorsToConsole);
```

Parameters

languageID

(IN) Specifies the language ID number for the language to be added (0-99 is reserved by the NetWare® OS.

newLanguageName

(IN) Specifies the name of the language to be added.

showErrorsToConsole

(IN) Specifies whether to show error messages to the console screen.

Return Values

0	Success
1	Invalid ID or name

Remarks

AddLanguage adds the new language name and ID number to the OS language list.

See Adding to OS Supported Language List: Example.

See Also

GetCurrentOSLanguageID, **LoadLanguageMessageTable**, **RenameLanguage**, **ReturnLanguageName**, **SetCurrentOSLanguageID**

GetCurrentOSLanguageID

Returns the language ID for the language currently running on the server

Local Servers: nonblocking

Remote Servers: N/A

Classification: 4.x

Platform: NLM

SMP Aware: No

Service: Internationalization

Syntax

```
#include <nwadv.h>

int GetCurrentOSLanguageID (
   void);
```

Return Values

Returns the language ID of the language for the OS.

See Also

AddLanguage, **LoadLanguageMessageTable**, **RenameLanguage**, **ReturnLanguageName**, **SetCurrentOSLanguageID**

LoadLanguageMessageTable

Loads a pointer to a language message table so the table can be used for language enabling support

Local Servers: nonblocking

Remote Servers: N/A

Classification: 4.x

Platform: NLM

SMP Aware: No

Service: Internationalization

Syntax

```
#include <nwadv.h>

LONG LoadLanguageMessageTable (
   char***messageTable,
   LONG  *messageCount,
   LONG  *languageID);
```

Parameters

messageTable

(IN/OUT) Points to an array of pointers containing the Language Module file created with the MsgTools utility.

messageCount

(OUT) Points to the number of messages in the *messageTable* parameter.

languageID

(OUT) Points to the language ID number of the language of the *messageTable* parameter.

Return Values

0	Successful
-1	Unsuccessful

Remarks

NLM™ applications can be enabled for multiple languages by calling **LoadLanguageMessageTable**. Default language tables can be bound to NLM applications with the MESSAGES option in NLMLINK or NLMLINKX.

See Also

AddLanguage, **GetCurrentOSLanguageID**, **RenameLanguage**, **ReturnLanguageName**, **SetCurrentOSLanguageID**

localeconv

Sets the components of an object with values appropriate for the formatting of numeric quantities according to the current locale

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Platform: NLM

SMP Aware: Yes

Service: Internationalization

Syntax

```
#include <locale.h>

struct lconv *localeconv (void);
```

Return Values

Returns a pointer to the filled-in object.

Remarks

localeconv sets the components of a struct lconv object with values appropriate for the formatting of numeric quantities according to the current locale. See the lconv structure.

See Also

NWLlocaleconv, **NWLsetlocale**, **setlocale**

NWatoi

Converts a string to an integer
NetWare Server: 4.x
Platform: DOS, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWatoi (
   pnstr   string);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWatoi
   (string : pnstr
) : nint;
```

Parameters

string
(IN) Points to the string of digits to be converted into an integer value.

Return Values

0x0000	No digit was converted
non-zero	Value converted from string

Remarks

On the server, this function is called "NWLatoi" and in a double-byte environment. It will return 0 if there are any double-byte characters in the string.

NWatoi coverts a string of digits into an integer value. The syntax of the string must be:

```
long ::= [isspace]*[sign]digit[digit]*
```

CAUTION: Only decimal integers will work with NWatoi.

Tabs and/or spaces can precede the digits to be converted. A plus (+) or minus (-) sign can immediately precede the digits to be converted. Otherwise, any non-digit character terminates the conversion. Currently no double-byte characters are converted.

If the string cannot be converted to an integer, 0x0000 will be returned.

NCP Calls

None

See Also

NWisdigit

NWCharLwr

Converts a character to lower case
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

nint N_API NWCharLwr (
   nint   chr);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWCharLwr
  (ch : nint
) : nint;
```

Parameters

chr
(IN) Specifies the character to be converted to lower case.

Return Values

Returns the lower case value of *chr*.

Remarks

Only single-byte characters are returned.

NCP Calls

None

See Also

NWCharUpr, **NWLsetlocale**, **NWLstrupr**

NWCharType

Determines whether a character is single- or double-byte

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWCharType (
   nint   ch);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWCharType
  (ch : nint
) : nint;
```

Parameters

ch

(IN) Indicates the character to be tested (one or two bytes).

Return Values

0x0001	NWSINGLE_BYTE
0x0002	NWDOUBLE_BYTE

NCP Calls

None

See Also

NWLsetlocale

NWCharUpr

Converts a character to uppercase
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint NWAPI NWCharUpr (
   nint   chr);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWCharUpr
  (ch : nint
) : nint;
```

Parameters

chr

(IN) Specifies the character to be converted to upper case.

Return Values

Returns the uppercase value of the *chr* parameter.

Remarks

Only single-byte characters are returned.

NCP Calls

None

See Also

NWCharLwr, **NWLsetlocale**, **NWLstrupr**

NWCharVal

Returns the integer value of a character or a character in a string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWCharVal (
   const nstr N_FAR  *string);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWCharVal
  (str : pnstr
) : nint;
```

Parameters

string
(IN) Points to a character in a string.

Return Values

Returns the integer value of character pointed to by the *string* parameter.

Remarks

NWCharVal is useful when comparing the values of two characters. Double-byte characters are handled correctly.

NCP Calls

None

See Also

NWLsetlocale

NWfprintf

Formats a string and outputs it to a data stream (parameter reordering is supported)

NetWare Server: 4.x

Platform: DOS

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY_C (nint) NWfprintf (
   FILE N_FAR         *stream,
   const nstr N_FAR   *format,
                      [parameter_1,
                       parameter_2]...);
```

Parameters

stream

(OUT) Points to the stream to receive formatted data.

format

(IN) Points to the format string determining how the data will be formatted before sending it to the stream.

parameter_1, parameter_2

(IN) Specifies the user-supplied variable list of parameters whose values are used in the formatted output.

Return Values

Returns the number of bytes output.

Remarks

This function is currently supported on DOS platforms only.

For information on data formatting, see the **printf** function in any C manual.

NWfprintf is the same as C standard **fprintf** function except that parameter reordering is supported.

Parameter reordering allows a language translator to modify the format string and output the parameters in a different order than is specified in the **fprintf** call.

The reordering feature allows a language translator to change the order in which arguments are printed by just changing the format string.

There are two ways to do parameter reordering. The most common way is to specify the argument order with a %n in front of each formatting code. For example:

```
NWprintf("There are %d files in the directory %s.\n",
numfiles, dirname);
```

In an internationalized program, the format string would actually be stored in a separate file to be translated into other languages. The following statement changes the order of the arguments with only a change in the format string:

```
NWprintf("Directory %2%s contains %1%d files.\n",
numfiles, dirname);
```

The second method allows parameter ordering to be determined at run time. The format string is prepended with a reordering vector specifying the argument order. The reordering vector is a series of bytes with the following format:

```
LDH!<n><o1><o2><o3>...<format string> <n><o1><o2><o3>
etc. are binary bytes.
```

Where:

<n> is the number of arguments in the printf statements. <o1> specifies which argument should be printed first. If the third argument should be printed first, then <o1> would contain the binary value 3. <o2> specifies which argument should be printed second, etc..

NCP Calls

None

See Also

NWsprintf, NWprintf, NWvfprintf, NWvsprintf, NWvprintf

NWGetCollateTable

Gets the Character Collation Table from the current OS
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWGetCollateTable (
   pnstr    retCollateTable,
   size_t   maxLen);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWGetCollateTable
  retCollateTable : pnstr;
  maxLen          : size_t
) : nint;
```

Parameters

retCollateTable

(IN) Points to the buffer receiving the collate table.

maxLen

(IN) Specifies the maximum buffer size.

Return Values

0x0000	Successful
non-zero	Operating System Error

Remarks

A collation table is a 256-byte array giving the collation weight for each character in the current code page. The collation weight indicates how characters should be sorted, which varies from country to country.

NCP Calls

None

See Also

NWLsetlocale

NWGetNWLOCALEVersion

Returns the library version

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

void N_API NWGetNWLOCALEVersion (
   pnuint8   majorVersion,
   pnuint8   minorVersion,
   pnuint8   revisionLevel,
   pnuint8   betaReleaseLevel);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWGetNWLOCALEVersion
  (majorVersion     : pnuint8;
   minorVersion     : pnuint8;
   revisionLevel    : pnuint8;
   betaReleaseLevel : pnuint8
);
```

Parameters

majorVersion

(OUT) Points to an ASCII string containing the NWLocale major version.

minorVersion

(OUT) Points to an ASCII string containing the NWLocale minor version.

revisionLevel

(OUT) Points to the revision level.

betaReleaseLevel

(OUT) Points to the beta level.

Return Values

None

NCP Calls

None

NWIncrement

Increments a multibyte string pointer by a specified number of characters

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWIncrement (
   pnstr    string,
   size_t   numChars);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWIncrement
  (str      : pnstr;
   numChars : size_t
) : pnstr;
```

Parameters

string

(IN) Points to the string to increment.

numChars

(IN) Specifies the number of characters to increment.

Return Values

NULL	Less than the *numChars* parameter
non-NULL	Pointer to the position of the *numChars* parameter past the starting point of the string

Remarks

NWIncrement increments by characters, not bytes.

NCP Calls

None

See Also

NWLsetlocale, **NWNextChar**

NWisalnum

Returns a non-zero value if the given character is a letter, a digit, or a double-byte character

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWisalnum (
   nuint   ch);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWisalnum
  (ch : nuint
) : nint;
```

Parameters

ch

(IN) Specifies the character to be checked to determine if it is an alphabetic or numeric character.

Return Values

0x0000	Not an alphabetic or numeric character
non-zero	An alphabetic or numeric character

Remarks

DBCS characters are treated as alphanumeric characters. All double-byte characters return a non-zero value.

Under an NLM, double-byte characters return 0.

NCP Calls

None

See Also

NWisalpha, **NWisdigit**

NWisalpha

Returns a non-zero value if the given character is a letter or DBCS
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWisalpha (
   nuint   ch);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWisalpha
  (ch : nuint
) : nint;
```

Parameters

ch

(IN) Specifies the character to be checked to determine if it is an alphabetic character.

Return Values

0x0000	Not an alphabetic character
non-zero	An alphabetic character

Remarks

All double-byte characters return a non-zero value.

Under an NLM, double-byte characters return 0.

NCP Calls

None

See Also

NWisalnum, **NWisdigit**

NWisdigit

Returns a non-zero value if the given character is a digit
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWisdigit (
  nuint   ch);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWisdigit
  (ch : nuint
) : nint;
```

Parameters

ch

(IN) Specifies the character to be checked to determine if it is a numeric character.

Return Values

0x0000	Not a numeric character
non-zero	A numeric character

Remarks

DBCS characters passed to **NWisdigit** should be byte-swapped to Intel* lo-hi order.

Currently all double-byte characters return zero.

NCP Calls

None

See Also

NWisalnum, **NWisalpha**, **NWisxdigit**

NWisxdigit

Returns a non-zero value if the given character is a hexadecimal digit
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

nint N_API NWisxdigit (
  nuint   ch);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWisxdigit
(ch : nint
) : nint;
```

Parameters

ch

(IN) Specifies the character to be checked to determine if it is a hexadecimal character.

Return Values

0x0000	Not a hexadecimal character
non-zero	A hexadecimal character

Remarks

DBCS characters passed to **NWisxdigit** should be byte-swapped to Intel* lo-hi order.

Currently all double-byte characters return zero.

NCP Calls

None

See Also

NWisalnum, **NWisalpha**, **NWisdigit**

NWitoa

Converts an integer to a string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

pnstr N_API NWitoa (
   nint   value,
   pnstr   string,
   nuint   radix);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWitoa
  (value : nint;
   str : pnstr;
   radix : nuint
) : pnstr;
```

Parameters

value

(IN) Specifies the integer to be converted.

string

(OUT) Points to the string result.

radix

(IN) Specifies the base of the *value* parameter (range 2-36).

Return Values

Returns a pointer to the string.

Remarks

NWitoa converts the digits of the specified *value* parameter to a NULL-terminated character string and stores the results in the*string* parameter.

If the *radix* parameter equals 10 and the *value* parameter is negative, the first character of the returned string is the minus sign. Otherwise, the value is treated as an unsigned value.

NWitoa returns NULL if an invalid value is passed into the *radix* parameter.

The *string* parameter must be pointing to a buffer large enough to contain the number being converted.

NCP Calls

None

See Also

NWltoa, **NWultoa**, **NWutoa**

NWLInsertChar

Inserts a character at a given position in a string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <nwlocale.h>

nint N_API NWLInsertChar (
   pnstr   src,
   pnstr   insertableChar);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWisxdigit
  (src : pnstr;
   insertableChar : pnstr
) : nint;
```

Parameters

src

(IN/OUT) Points to a NULL-terminated string where the character is to be inserted.

insertableChar

(IN) Points to the character to be inserted (single- or double-byte).

Return Values

Returns the size of the inserted character (1 or 2 bytes).

NCP Calls

None

NWLlocaleconv

Sets the elements of the LCONV structure according to the current locale setting

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

LCONV N_FAR *N_API NWLlocaleconv (
  LCONV N_FAR   *lconvPtr);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLlocaleconv
  (lconvPtr : pLCONV
) : pLCONV;
```

Parameters

lconvPtr

(OUT) Points to the LCONV structure.

Return Values

Returns a pointer to the LCONV structure.

Remarks

Call **NWLlocaleconv** to get locale information to complete tasks not provided for by other Internationalization functions. **NWLlocaleconv** sets the elements in the specified structure according to the currently selected locale.

NWLlocaleconv is not an exact analog to the ANSI **localeconv** function because **NWLlocaleconv** contains information that is not ANSI-specific. Also, ANSI character pointers have been changed to character arrays, which is required for Windows 3.x-4.x in the enhanced mode.

For a more detailed description of the ANSI elements in the above structure, refer to the ANSI specification, *ANSI X3.159-1989*.

NCP Calls

None

See Also

localeconv, **NWLsetlocale**

NWLmblen

Returns the length of a multibyte character
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

nint N_API NWLmblen (
   pnstr    string,
   size_t   maxBytes);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLmblen
   (string : pnstr;
    count : size_t
) : nint;
```

Parameters

string

(IN) Points to the string containing a character to evaluate.

count

(IN) Specifies the maximum number of bytes to evaluate.

Return Values

If the *string* parameter is not NULL, returns the length (in bytes) of the multibyte character.

If the *string* parameter is NULL, returns zero.

If the string does not form a valid multibyte character within the first count characters, returns -1.

Remarks

Generally, *maxBytes* is set to 2 and the function returns 1 for a 1-byte or 2 for a 2-byte character.

Use the **NWLmbslen** function to count the number of character is a string.

NCP Calls

None

See Also

NWLmbslen

NWLmbslen

Returns the number of characters (not bytes) in a specified string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

N_EXTERN_LIBRARY (size_t) NWLmbslen (
   const nuint8  *string);
```

Parameters

string
(IN) Specifies a NULL-terminated string.

Return Values

Returns the number of characters in the string.

Remarks

The string may consist of both single- and double-byte characters.

NCP Calls

None

See Also

NWLmblen, **NWstrlen**

NWLsetlocale

Initializes the implementation-defined native locale
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWLsetlocale (
   nint               category,
   const nstr N_FAR  *locale);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLsetlocale
  (category     : nint;
   const locale : pnstr
) : pnstr;
```

Parameters

category

(IN) Specifies the locale categories.

locale

(IN) Points to a locale string.

Return Values

NULL	Failure
non-NU LL	Pointer to the Country ID

Remarks

WARNING: Call NWLsetlocale before calling any locale-sensitive functions. Typically NWLsetlocale is called with the following

parameters:

```
NWLsetlocale(LC_ALL,"");
```

After calling **NWLsetlocale**, all locale-sensitive functions will use the locale information set by **NWLsetlocale**.

locale can have the following values:

```
Initialize the implementation-defined native
         environment.
NULL     Query for the current locale country ID,
         without initializing the environment.
```

The *category* parameter and internal data structures can be specified and intialized for that category only using less initialization time. Functions affected by an uninitialized category are not called.

The *category* parameter can have the following values:

```
NLC_ALL          0
NLC_COLLATE      1
NLC_CTYPE        2
NLC_MONETARY     3
NLC_NUMERIC      4
NLC_TIME         5
NLC_TOTAL         6
```

The country ID is a three-digit string defined by IBM. These IDs are based on the international phone prefix for a given country. For example, USA is `001'. Finland is `358'. The **NWCallsInit** function will automatically call the functions, **NWSetLocale** and **NWInitUnicodeTables**.

NCP Calls

None

See Also

NWInitUnicodeTables, **NWLlocaleconv**

NWLstrbcpy

Copies a locale-sensitive string into the target buffer and NULL-terminates the target string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY(pnstr) NWLstrbcpy (
   pnstr              dest,
   const nstr N_FAR  *src,
   size_t             maxlen);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrbcpy
  (dest : pnstr;
   const src : pnstr;
   maxlen: size_t
) : pnstr;
```

Parameters

dest

(OUT) Points to the string to copy to.

src

(IN) Points to the string to be copied.

maxlen

(IN) Specifies the maximum number of bytes that may be copied into the target buffer including the NULL terminator.

Return Values

NULL	The *dest* parameter and/or the *src* parameter is NULL
non-NULL	Pointer to the *dest* parameter

Remarks

NWLstrbcpy is similar to the ANSI standard **strncpy** function, except **NWLstrbcpy** will not pad the remaining space in the output buffer with zeroes as the **strncpy** function does.

NWLstrbcpy is similar to the **NWstrncpy** function, except **NWLstrbcpy** specifies the maximum bytes (not characters) to copy. **NWLstrbcpy** will also NULL-terminate the string.

If the source string is greater than or equal to the *maxlen* parameter in length, only the first n-1 bytes are copied to the target buffer followed by NULL.

NWLstrbcpy will not truncate a double-byte character. If the output buffer is not large enough to hold both bytes of a double-byte character, the string is terminated and neither byte will be copied.

NCP Calls

None

See Also

NWLsetlocale, **NWstrncpy**

NWLstrchr

Finds a character in a string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWLstrchr (
   pnstr   string,
   nint    find);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrchr (
   str  : pnstr;
   find : nint
) : pnstr;
```

Parameters

string

(IN) Points to the string to search.

find

(IN) Specifies the character for which to search (single- or double-byte).

Return Values

NULL	Character not found
non-NU LL	Pointer to the character found

Remarks

NWLstrchr is case-sensitive.

NCP Calls

None

See Also

NWLsetlocale, **NWLstrrchr**, **NWLstrstr**

NWLstrcoll

Compares two strings according to the rules of the current locale
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWLstrcoll (
   pnstr   string1,
   pnstr   string2);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrcoll
  (string1 : pnstr;
   string2 : pnstr
) : nint;
```

Parameters

string1

(IN) Points to the first string to compare.

string2

(IN) Points to the second string to compare.

Return Values

When the *string1*/*string2* parameter combination is relative to current locale setting:

integer>0	*string1*>*string2*
integer=0	*string1*=*string2*
integer<0	*string1*<*string2*

Remarks

NWLstrcoll compares `string1' to `string2'. Both are interpreted as appropriate to the LC_COLLATE category of the current locale. For a locale-sensitive comparison of two strings, call **NWLstrcoll** instead of the **strcmp** function.

The comparison between string1 and string 2 is not case sensitive.

NCP Calls

None

See Also

NWLsetlocale, **NWLstricmp**

NWLstrcspn

Computes the segment length of a specified string containing characters not found in another string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY(size_t) NWLstrcspn (
   const nstr N_FAR  *string1,
   const nstr N_FAR  *string2);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrcspn
  (const string1 : pnstr;
   const string2 : pnstr
) : size_t;
```

Parameters

string1

(IN) Points to the string to examine for characters from the*string2* parameter.

string2

(IN) Points to the characters to look for in the*string1* parameter.

Return Values

0x0000	*string1*=0
non-zero	0-based byte position in the*string1* parameter of the first character which matches any character in the *string2* parameter

Remarks

NWLstrcspn computes the length (in bytes) of the maximum initial segment of the *string1* parameter consisting entirely of characters not in the *string2* parameter.

Either or both of the strings can have double-byte characters.

NCP Calls

None

See Also

NWLstrspn, **NWLstrpbrk**

NWLstrftime

Formats the time and date according to a specified format

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h>
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY(size_t) NWLstrftime (
   pnstr                  dst,
   size_t                 max,
   const nstr N_FAR      *fmt,
   const struct tm N_FAR  *ptm);
```

Parameters

dst

(OUT) Points to the buffer to contain date and time (if the *string* parameter is NULL, the length of the time string is returned).

max

(IN) Specifies the maximum size of the buffer where the date and time string is placed.

fmt

(IN) Points to the specified format.

ptm

(IN) Points to the *tm* structure.

Return Values

0x0000	The *max* parameter is 0 or the *fmt* parameter is NULL
non-zero	Length of formatted time

Remarks

NWLstrftime places characters into the *dst* parameter as formatted by the *fmt* parameter.

The *fmt* parameter can have the following values:

`%a`	`Abbreviated weekday name ("Mon")`
`%A`	`Full weekday name ("Monday")`
`%b`	`Abbreviated month name ("Feb")`
`%B`	`Full month name ("February")`
`%c`	`Date and time representation appropriate for locale ("2/22/97 04:30:00 pm" for USA locale)`
`%C`	`Century number ("19" if year is 1997)`
`%d`	`Day of month as 2-digit decimal number (01-31)`
`%D`	`Date in mm/dd/yy format ("02/22/97")`
`%e`	`Day of month as a 2-character space-padded number ("9" for the 9th day of the month)`
`%h`	`Abbreviated month name (same as %b)`
`%H`	`Hour in 2-digit 24-hour format (00-23)`
`%I`	`Hour in 2-digit 12-hour format (01-12)`
`%j`	`Day of year as 3-digit decimal number (001-366)`
`%m`	`Month as 2-digit decimal number (01-12)`
`%M`	`Minute as 2-digit decimal number (00-59)`
`%n`	`Newline character (/n)`
`%p`	`Current locales A.M./P.M. indicator for 12-hour clock ("am" or "pm" for USA locale)`
`%r`	`Current 12-hour locales AM/PM time format ("04:30:00 pm" for USA lo`
`%R`	`Time in HH:MM format ("16:30")`
`%S`	`Second as 2-digit decimal number (00-59)`
`%t`	`Tab character (/t)`
`%T`	`Time in HH:MM:SS format ("16:30:00")`
`%u`	`Weekday as decimal number (1-7; Monday is 1)`
`%U`	`Week of year as 2-digit decimal number with Sunday as first day of week (00-51)`
`%V`	`Week of year as 2-digit decimal number with Monday as first day of week; if the first day of the year is either Saturday or Sunday, ea treated as part of the previous year so the new year begins on Mond`
`%w`	`Weekday as decimal number (0-6; Sunday is 0)`
`%W`	`Week of year as 2-digit decimal number (same as %V)`
`%x`	`Date representation for current locale ("02/22/97" for USA locale)`
`%X`	`Time representation for current locale; may be in 12-hour format ac locales format ("04:30:00 pm" for USA locale)`
`%y`	`Year without century as 2-digit decimal number (00-99)`
`%Y`	`Year with century as 4-digit decimal number ("1997")`

If a weekday value is passed into **NWLstrftime** is beyond the 0-6 range, **NWLstrftime** will compute it.

If the *dst* parameter is set to NULL, the length of the formatted string is still returned. The formatted string is not returned.

NCP Calls

None

See Also

NWLsetlocale

NWLstricmp

Performs a case-insensitive compare of two strings
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

N_EXTERN_LIBRARY(nint) NWLstricmp (
   const nstr N_FAR  *str1,
   const nstr N_FAR  *str2);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWisxdigit
  (const str1 : pnstr;
   const str2 : pnstr
) : nint;
```

Parameters

str1

(IN) Points to the first string to compare.

str2

(IN) Points to the second string to compare.

Return Values

<0	If the *str1* parameter is less than the *str2* parameter.
=0	If the *str1* parameter is identical to the *str2* parameter.
>0	If the *str1* parameter is greater than the *str2* parameter.

Remarks

NWLstricmp compares uppercase versions of the *str1* and *str2* parameters and returns a value indicating their relationship.

NWLstricmp

NWLstricmp performs the comparison lexicographically using the code page value of the characters and does not use the locale sensitive collation sequence as the **NWLstrcoll** function. However, **NWLstricmp** is faster than the **NWLstrcoll** function and is efficient for comparing strings for equality.

NCP Calls

None

See Also

NWLstrcoll, **NWstrncoll**

NWLstrlwr

Converts a string to lower case using locale information
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

pnstr N_API NWLstrlwr (
   pnstr   string);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrlwr
  (str1 : pnstr
) : pnstr;
```

Parameters

string
(IN/OUT) Points to the string to be converted to lower case.

Return Values

Returns a pointer to the lower case*string* parameter.

Remarks

NWLstrlwr is sensitive to double-byte characters if the locale includes double-byte characters.

NCP Calls

None

See Also

NWCharLwr, **NWLsetlocale**, **NWLstrupr**

NWLstrpbrk

Locates the first occurrence in a specified string of given characters from another string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWLstrpbrk (
   pnstr              string1,
   const nstr N_FAR  *string2);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrpbrk
  (string1       : pnstr;
   const string2 : pnstr
) : pnstr;
```

Parameters

string1

(IN) Points to the string to search.

string2

(IN) Points to the list of characters to search for in the *string1* parameter.

Return Values

NULL	No characters from the *string2* parameter are in the *string1* parameter
non-NULL	Pointer to the first matching character (indicates characters from the *string2* parameter are in the *string1* parameter)

Remarks

NWLstrpbrk is sensitive to double-byte characters if the locale includes double-byte characters.

NCP Calls

None

See Also

NWLsetlocale, **NWLstrcspn**, **NWLstrspn**

NWLstrrchr

Locates the last occurrence of a character in a string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWLstrrchr (
   pnstr   string,
   nint    find);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrrchr
  (str  : pnstr;
   find : nint
) : pnstr;
```

Parameters

string
(IN) Points to the string to search.

find
(IN) Specifies the character for which to search (single- or double-byte).

Return Values

NULL	The *string* parameter is NULL or no match is found
non-NU LL	Pointer to the character specified by the *find* parameter (indicates a match is found)

Remarks

NWLstrrchr searches backwards for the character specified in the *find* parameter.

NCP Calls

None

See Also

NWLsetlocale, **NWLstrchr**

NWLstrrev

Reverses the order of the characters in a string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

psntr N_API NWLstrrev (
   pnstr   string1,
   pnstr   string2);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrrev
  (string1 : pnstr;
   string2 : pnstr
) : pnstr;
```

Parameters

string1

(OUT) Points to the reversed string.

string2

(IN) Points to the string to be reversed.

Return Values

Pointer to the reversed string.

Remarks

Double-byte characters are treated as single units.

NCP Calls

None

See Also

NWLsetlocale

NWLstrspn

Computes the segment length of a specified string consisting entirely of characters from another string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY(size_t) NWLstrspn (
   const nstr N_FAR  *string1,
   const nstr N_FAR  *string2);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrspn
  (const string1 : pnstr;
   const string2 : pnstr
) : size_t;
```

Parameters

string1

(IN) Points to the string to search for characters from the*string2* parameter.

string2

(IN) Points to the characters to look for in the*string1* parameter.

Return Values

0x0000	The *string1* or *string2* parameter is NULL
non-zero	0-based position of first non-matching character

Remarks

NWLstrspn computes the length (in bytes) of the maximum initial segment of the *string1* parameter consisting entirely of characters from the *string2* parameter.

Double-byte characters are treated as single units.

NCP Calls

None

See Also

NWLsetlocale, **NWLstrcspn**, **NWLstrpbrk**

NWLstrstr

Searches a specified string for a given character string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWLstrstr (
   pnstr   string,
   pnstr   searchString);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrstr
   (str : pnstr;
    searchString : pnstr
) : pnstr;
```

Parameters

string

(IN) Points to the string to be searched.

searchString

(IN) Points to the character string for which to search.

Return Values

NULL	If the string pointed to by the *searchString* parameter is not found within the *string* parameter, or the *searchString* parameter is NULL or an empty string
non-NULL	Points to the *string* parameter beginning with the characters specified in the *searchString* parameter

Remarks

Double-byte characters are handled properly.

Unlike the ANSI version, if the *searchString* parameter points to an empty string (has zero length), **NWLstrstr** will return NULL instead of a pointer to the *string* parameter.

NCP Calls

None

See Also

NWLsetlocale, **NWLstrchr**

NWLstrtok

Finds the next token in a specified string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

pnstr N_API NWLstrtok (
   pnstr   parse,
   pnstr   delim);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrtok
  (parse : pnstr;
   delim : pnstr
) : pnstr;
```

Parameters

parse

(IN/OUT) Points to the string containing the token(s).

delim

(IN) Points to the string containing delimiters (can include double-byte characters).

Return Values

Returns a pointer to the next token found in the *parse* parameter.

Returns NULL when no more tokens are found.

Remarks

Every time **NWLstrtok** is called, it modifies the *parse* parameter by substituting a NULL character for each delimiter that is found.

Sequentially calling **NWLstrtok** breaks the string pointed to by the *parse* parameter into a sequence of tokens, each of which is delimited by a character from the string pointed to by the *delim* parameter. The first time

NWLstrtok is called in the sequence, the *parse* parameter is its first parameter. It will be followed by calling **NWLstrtok** additional times with a NULL pointer as the first parameter. The separator string pointed to by the *delim* parameter might be different each time**NWLstrtok** is called.

The first time in the sequence that **NWLstrtok** is called, it searches the string pointed to by the *parse* parameter for the first character that is NOT contained in the current separator string pointed to by the *delim* parameter. If no such character is found, there are no tokens in the string and **NWLstrtok** returns a NULL pointer. If such a character is found, the character is the start of the first token.

NWLstrtok continues searching, beginning at the token found for a character that IS contained in the current separator string. If no such character is found, the current token extends to the end of the string pointed to by the *parse* parameter. Subsequent searches for a token will return a NULL pointer. If such a character is found, it is overwritten by a NULL character, which terminates the current token. **NWLstrtok** saves a pointer to the following character, from which the next search for a token will start.

Each subsequent call, with a NULL pointer as the value of the first parameter, starts searching from the saved pointer as described.

WARNING: NWLstrtok uses a static variable for maintaining the last next token location. If multiple or simultaneous calls are made to NWLstrtok, a high potential for data corruption and incorrect results exists.

Do not attempt to call NWLstrtok simultaneously for different strings. Be aware of calling NWLstrtok from within a loop where another application might also be calling NWLstrtok.

NCP Calls

None

See Also

NWLstrspn, **NWLstrcspn**

NWLstrupr

Converts a string to uppercase using locale information
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWLstrupr (
   pnstr   string);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrupr
  (str : pnstr
) : pnstr;
```

Parameters

string
(IN/OUT) Points to the string to be converted to uppercase.

Return Values

Returns a pointer to the lower case*string* parameter.

Remarks

NWLstrupr is sensitive to double-byte characters if the locale includes double-byte characters.

NCP Calls

None

See Also

NWCharUpr, **NWLsetlocale**, **NWLstrlwr**

NWLstrxfrm

Transforms a string by replacing each character with its corresponding collation value

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

size_t N_API NWLstrxfrm (
   pnstr    string1,
   pnstr    string2,
   size_t   numBytes);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWLstrxfrm
  (string1  : pnstr;
   string2  : pnstr;
   numBytes : size_t
) : size_t;
```

Parameters

string1

(OUT) Points to the string after it is transformed to its collation value (optional).

string2

(IN) Points to the string to be transformed.

numBytes

(IN) Specifies the size of the output buffer in bytes (including the NULL terminator).

Return Values

0x0000	String not transformed
non-zero	Length of the complete transformed string (not just the part

	of the string that fits in the output buffer).

Remarks

NWLstrxfrm transforms the *string2* parameter to the corresponding collation weights and places the results in the *string1* parameter.

Only the number of bytes specified by the *numBytes* parameter will be written to the *string1* parameter. If the *numBytes* parameter is zero, the *string1* parameter can be NULL.

To find the total number of bytes required to hold the transformed string and the NULL terminator, use the expression

```
(NWLstrxfrm(NULL, input, 0) +1)
```

If the return value plus one is greater than the number specified by the *numBytes* parameter, the transformed string did not fit in the output buffer and the output buffer might not be NULL terminated. If the return value plus one is less than or equal to the number specified by the *numBytes* parameter, the entire string (including the NULL terminator) is contained in the output buffer.

A performance advantage is provided by calling **NWLstrxfrm** in place of the **NWLstrcoll** function when multiple comparisons of the same string are necessary. If you call the **strcmp** function with two transformed strings, the result is similar to calling the **NWLstrcoll** function with the two original strings. By calling **NWLstrxfrm**, however, only the first byte of a double-byte character is used to determine the collation weight, while the **NWLstrcoll** function compares both bytes.

If the collation table is not loaded, the *numBytes* parameter is `0', or the *string2* parameter is NULL, zero will be returned.

Under Windows 3.1, the length of the input string (including NULL) is limited to 256 bytes.

NOTE: **NWLstrxfrm** does not distinguish between characters with accents and characters without accents unless the character is specifically in the alphabet for the country.

NCP Calls

None

See Also

NWGetCollateTable **NWLsetlocale**, **NWLstrcoll**

NWltoa

Converts a long integer to a string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

pnstr N_API NWltoa (
   nuint32   value,
   pnstr     buf,
   nuint     radix);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWltoa
  (value : nint32;
   buf : pnstr;
   radix : nuint
) : pnstr;
```

Parameters

value

(IN) Specifies the long integer to be converted.

buf

(OUT) Points to a buffer to hold the string result.

radix

(IN) Specifies the base of the *value* parameter (range 2-36).

Return Values

Returns a pointer to the string.

Remarks

NWltoa converts the digits of the specified *value* parameter to a NULL-terminated character string and stores the results in the *buf* parameter.

If the *radix* parameter equals 10 and the *value* parameter is negative, the first character of the returned string is the minus sign. Otherwise, the value is treated as an unsigned value.

NWltoa returns NULL if an invalid value is passed into the *radix* parameter.

The *buf* parameter must be pointing to a buffer large enough to contain the number being converted.

NCP Calls

None

See Also

NWitoa, **NWultoa**, **NWutoa**

NWLTruncateString

Truncates a string at the specified number of bytes
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

N_EXTERN_LIBRARY (nint) NWLTruncateString (
   pnchar8   pStr,
   nint      iMaxLen);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWltoa
  (pStr : pnchar8;
   iMaxLen : nint
) : nint;
```

Parameters

pStr

(IN) Points to the first character in the string (can contain double-byte characters).

iMaxLen

(IN) Specifies the maximum length of the string, including the NULL terminator, in bytes.

Return Values

Returns the length of the string (might be truncated) not counting the NULL terminator (in bytes).

Remarks

NWLTruncateString truncates the string if necessary so the entire string, including the NULL termination byte, will fit in a buffer of the number of bytes specified by the *iMaxLen* parameter.

If the truncation would chop a double-byte character in half, the first half of the double-byte character is also eliminated.

NCP Calls

None

NWNextChar

Increments a pointer to the next character in a string with multibyte characters

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWNextChar (
   const nstr N_FAR  *string);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWNextChar
  (const str : pnstr
) : pnstr;
```

Parameters

string

(IN) Points to the address of the current position in the string. If there is no next character, *string* points to the null character at the end of the string.

Return Values

Pointer to the next character (not byte) in the specified string.

Remarks

NWNextChar is called to move through strings whose characters are one or two bytes each in length. For example, **NWNextChar** could be called for strings containing characters from a Japanese character set.

In a multibyte string, it is not obvious whether the current byte is a single-byte character or the second character of a double-byte character. **NWNextChar** resolves this ambiguity.

Increment a pointer witht he statement `ptr=NWNextChar(ptr)` instead of `ptr++`.

The **NWNextChar** function is implemented as a call to **AnsiNext** in Windows.

NCP Calls

None

See Also

NWLsetlocale, **NWIncrement**, **NWPrevChar**

NWPrevChar

Finds the beginning of the nearest previous character in a string with multibyte characters

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWPrevChar (
   const nstr N_FAR  *string,
   pnstr              position);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWPrevChar
  (const str : pnstr;
   position : pnstr
) : pnstr;
```

Parameters

string

(IN) Points to the beginning of the string.

position

(IN) Points to a character in the NULL-terminated string.

Return Values

0x0000	The *string* parameter is NULL
non-zero	Pointer to the previous character in the string. If the *position* parameter is equal to the *string* parameter, this value points to the first character in the string

Remarks

In a multibyte string, it is not obvious whether the current byte is a single-byte character or the second character of a double-byte character. **NWPrevChar** resolves this ambiguity.

NCP Calls

None

See Also

NWLsetlocale, **NWNextChar**

NWprintf

Formats and outputs a string to stdout (parameter reordering is supported)
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY_C(nint) NWprintf (
   const nstr N_FAR  *format,
                     [parameter_1,
                      parameter_2]...);
```

Parameters

format

(IN) Points to the format string determining how the data will be formatted before sending it to standard output.

parameter_1, parameter_2, ...

(IN) Specifies the user-supplied variable list of parameters whose values are used in the formatted output.

Return Values

Returns the number of bytes output.

Remarks

The **NWprintf** function is the same as the standard C printf with the exception that it supports parameter reordering. This feature allows a language translator to change the order in which arguments are printed by just changing the format string.

There are two ways to do parameter reordering. The most common way is to specify the argument order with a %n in front of each formatting code. For example:

```
NWprintf("There are %d files in the directory %s.\n",
numfiles, dirname);
```

In an internationalized program, the format string would actually be stored in a separate file to be translated into other languages. The following statement changes the order of the arguments with only a change in the format string:

```
NWprintf("Directory %2%s contains %1%d files.\n",
numfiles, dirname);
```

The second method allows parameter ordering to be determined at run time. The format string is prepended with a reordering vector specifying the argument order. The reordering vector is a series of bytes with the following format:

```
LDH!<n><o1><o2><o3>...<format string> <n><o1><o2><o3>
etc. are binary bytes.
```

Where:

<n> is the number of arguments in the printf statements. <o1> specifies which argument should be printed first. If the third argument should be printed first, then <o1> would contain the binary value 3. <o2> specifies which argument should be printed second, etc..

NCP Calls

None

See Also

NWsprintf, **NWfprintf**, **NWvfprintf**, **NWvsprintf**, **NWvprintf**

NWsprintf

Formats a string and outputs it to a user buffer (parameter reordering is supported)

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY_C(nint) NWsprintf (
   pnstr               buffer,
   const nstr N_FAR   *format,
                      [parameter_1,
                       parameter_2]...);
```

Parameters

buffer

(OUT) Points to the buffer receiving the formatted data.

format

(IN) Points to the format string determining how the data will be formatted before sending it to the buffer.

parameter_1, parameter_2, ...

(IN) Specifies the user-supplied variable list of parameters whose values are used in the formatted output.

Return Values

Returns the number of bytes output.

Remarks

For information about data formatting, see the **printf** function in any C manual.

If the **NWsprintf** function is the same as the C standard **sprintf** function except that parameter reordering is supported.

The reordering feature allows a language translator to change the order in which arguments are printed by just changing the format string.

There are two ways to do parameter reordering. The most common way is to specify the argument order with a %n in front of each formatting code. For example:

```
NWprintf("There are %d files in the directory %s.\n",
numfiles, dirname);
```

In an internationalized program, the format string would actually be stored in a separate file to be translated into other languages. The following statement changes the order of the arguments with only a change in the format string:

```
NWprintf("Directory %2%s contains %1%d files.\n",
numfiles, dirname);
```

The second method allows parameter ordering to be determined at run time. The format string is prepended with a reordering vector specifying the argument order. The reordering vector is a series of bytes with the following format:

```
LDH!<n><o1><o2><o3>...<format string> <n><o1><o2><o3>
etc. are binary bytes.
```

Where:

<n>   is the number of arguments in the printf statements. <o1> specifies which argument should be printed first. If the third argument should be printed first, then <o1> would contain the binary value 3. <o2> specifies which argument should be printed second, etc..

NCP Calls

None

See Also

NWfprintf, **NWprintf**, **NWvfprintf**, **NWvsprintf**, **NWvprintf**

NWstrImoney

Returns the country prefix and money format for a numerical value
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWstrImoney (
   pnstr         buffer,
   NUMBER_TYPE   Value);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWstrImoney
  (buffer : pnstr;
   Value : NUMBER_TYPE
) : pnstr;
```

Parameters

buffer
(OUT) Points to the number formatted for a specific country in the international format.

Value
(IN) Specifies the number to format.

Return Values

Returns a pointer to *string*.

Remarks

NWstrImoney has no ANSI counterpart.

The last 2 or 3 (some locales) digits of the *Value* parameter are always formatted as the smallest money units for that locale. The *Value* parameter is of the nint32 type.

NWstrImoney is different from the **NWstrmoney** function since it provides country prefixes in the *buffer* parameter.

The *buffer* parameter can have the following values:

BELGIUM	BEF
CANADA_FR	CAD
DENMARK	DKK
FINLAND	FIM
FRANCE	FRF
GERMANY	DDM
ITALY	ITL
NETHERLANDS	NLG
NORWAY	NOK
PORTUGAL	PTE
SPAIN	ESP
SWEDEN	SEK
SWITZERLAND	SFR
UK	GBP
USA	USD
JAPAN	JPY
KOREA	KRW
PRC	CNY
TAIWAN	TWD

Examples:

locale	**number**	**formatted value**
US	1234	USD 12.34
FRANCE	1234	FRF 12,34

For example, if country code 033 (France) were used and 3498 is passed in the *Value* function, the output string would be "FRF 34,98".

NCP Calls

None

See Also

NWLsetlocale, **NWstrmoney**

NWstrlen

Returns the number of bytes in a string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

N_EXTERN_LIBRARY(nint) NWstrlen (
   const nstr N_FAR  *str);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWstrlen
  (const str : pnstr
) : nint;
```

Parameters

str
(IN) Points to the NULL-terminated string to be counted.

Return Values

Returns the number of bytes (not characters) in a string (not including the NULL-termination byte).

NCP Calls

None

See Also

NWLmbslen

NWstrmoney

Returns the locale-sensitive money format for a numerical value
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWstrmoney (
   pnstr          buffer,
   NUMBER_TYPE    Value);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWstrmoney
  (buffer : pnstr;
   Value : NUMBER_TYPE
) : pnstr;
```

Parameters

buffer

(OUT) Points to the number formatted for a specific country.

Value

(IN) Specifies the number to format.

Return Values

Returns a pointer to *string*.

Remarks

NWstrmoney has no ANSI counterpart. **NWstrmoney** is different from **NWstrImoney**; it does not provide the country prefix in *string*.

The last 2 or 3 (some locales) digits of *Value* will always be formatted as the smallest money units for that locale. The type of this variable is long double.

Examples:

locale	**number**	**formatted value**
US	1234	$12.34
FRANCE	1234	p12,34

NCP Calls

None

See Also

NWLsetlocale, **NWstrImoney**

NWstrncoll

Returns the locale-sensitive comparison of two strings

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

nint N_API NWstrncoll (
   pnstr    string1,
   pnstr    string2,
   size_t   maxBytes);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWstrncoll
  (string1  : pnstr;
   string2  : pnstr;
   maxBytes : size_t
) : nint;
```

Parameters

string1

(IN) Points to the first string to compare.

string2

(IN) Points to the second string to compare.

maxBytes

(IN) Specifies the maximum number of bytes to compare.

Return Values

0x0000	Two strings are identical or the *maxBytes* parameter is zero
<0	The *string1* parameter is less than the *string2* parameter, relative to the current locale setting
>0	The *string1* parameter is greater than the *string2* parameter,

	relative to the current locale setting

Remarks

If no collate table exists, the locale is the C locale (see the **strcmp** function).

NWstrncoll is useful to insert or delete items in a sorted list based on their collation value.

Double byte characters count as one character in the count.

The comparison between the *string1* and *string2* parameters is not case sensitive.

NCP Calls

None

See Also

NWLsetlocale, **NWLstrcoll**, **NWLstrxfrm**

NWstrncpy

Copies a locale-sensitive string for a specified number of characters (not bytes)

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWstrncpy (
   pnstr   target_string,
   pnstr   source_string,
   nint    numChars);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWstrncpy
  (target_string : pnstr;
   source_string : pnstr;
   numChars      : nint
) : pnstr;
```

Parameters

target_string

(OUT) Points to the string to which to copy.

source_string

(IN) Points to the string to be copied.

numChars

(IN) Specifies the number of characters to copy.

Return Values

NULL	The *target_string* parmaeter and/or the *source string* parameter is NULL
non-NULL	Pointer to the *target_string* parameter

Remarks

NWstrncpy copies a string of up to the number of characters specified by the *numChars* parameter to the destination string and NULL-terminates the string if the source string contains less characters than the number specified by the *numChars* parameter. However, unlike the ANSI **strncpy** function, **NWstrncpy** does not pad any remaining space in the destination string with NULLs.

To guarantee a NULL-terminated string, set the *numChars* parameter equal to the length of the *target_string* parameter minus one and set the last element of the *target_string* parameter equal to zero.

NCP Calls

None

See Also

NWLsetlocale

NWstrnum

Formats a number for a specific country and returns the number in a string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

pnstr N_API NWstrnum (
   pnstr          buffer,
   NUMBER_TYPE    Value);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWstrnum
  (buffer : pnstr;
   Value  : NUMBER_TYPE
) : pnstr;
```

Parameters

buffer

(OUT) Points to the number formatted for a specific country.

Value

(IN) Specifies the number to format (long).

Return Values

Returns a pointer to the*string* parameter.

Remarks

If country code 033 (France) was used and 3498 was passed in the *Value* parameter, **NWStrnum** will return 3.498.

Examples:

locale	**number**	**formatted value**

```
US          1234            1,234
FRANCE      1234            1.234
```

NCP Calls

None

See Also

NWLsetlocale

NWultoa

Converts a long unsigned integer to a string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <nwlocale.h>

pnstr N_API NWultoa (
   nuint32   value,
   pnstr     buf,
   nuint     radix);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWultoa
  (value : nuint32;
   buf : pnstr;
   radix : nuint
) : pnstr;
```

Parameters

value

(IN) Specifies the long unsigned integer to be converted.

buf

(OUT) Points to the string result.

radix

(IN) Specifies the base of the *value* parameter (range 2-36).

Return Values

Returns a pointer to the string.

Remarks

NWultoa converts the digits of the specified *value* parameter to a NULL-terminated character string and stores the results in the *buf* parameter.

NWultoa returns NULL if an invalid value is passed into the *radix* parameter.

The *buf* parameter must be pointing to a buffer large enough to contain the number being converted.

NCP Calls

None

See Also

NWitoa, **NWltoa**, **NWutoa**

NWutoa

Converts an unsigned integer to a string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Internationalization

Syntax

```
#include <nwlocale.h>

pnstr N_API NWutoa (
   nuint   value,
   pnstr   string,
   nuint   radix);
```

Pascal Syntax

```
#include <nwlocale.inc>

Function NWultoa
  (value : nuint32;
   str : pnstr;
   radix : nuint
) : pnstr;
```

Parameters

value
(IN) Specifies the unsigned integer to be converted.

string
(OUT) Points to the string result.

radix
(IN) Specifies the base of the *value* parameter (range 2-36).

Return Values

Returns a pointer to the string.

Remarks

NWutoa converts the digits of the specified *value* parameter to a NULL-terminated character string and stores the results in the*string* parameter.

NWutoa returns NULL if an invalid value is passed into the *radix* parameter.

The *string* parameter must be pointing to a buffer large enough to contain the number being converted.

NCP Calls

None

See Also

NWitoa, **NWltoa**, **NWultoa**

NWvfprintf

Formats a string and outputs it to a data stream (parameter reordering is supported)

NetWare Server: 4.x

Platform: DOS

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY(nint) NWvfprintf (
   FILE N_FAR        *stream,
   const nstr N_FAR  *format,
   va_list            argList);
```

Parameters

stream

(IN) Points to the stream to receive formatted data.

format

(IN) Points to the format string determining how the data will be formatted before sending it to the stream.

argList

(IN) Points to a variable list of parameters whose values are used in the formatted output.

Return Values

Returns the number of bytes output.

Remarks

The variable argument list is given as a single va_list parameter.

For information about formatting data, see the **printf** function in any C manual.

Note that the **NWvfprint** function is currently supported on DOS platforms only. This function is the same as the C standard **vfprintf** function except that parameter reordering is supported.

The reordering feature allows a language translator to change the order in which arguments are printed by just changing the format string.

There are two ways to do parameter reordering. The most common way is to specify the argument order with a %n in front of each formatting code. For example:

```
NWprintf("There are %d files in the directory %s.\n",
numfiles, dirname);
```

In an internationalized program, the format string would actually be stored in a separate file to be translated into other languages. The following statement changes the order of the arguments with only a change in the format string:

```
NWprintf("Directory %2%s contains %1%d files.\n",
numfiles, dirname);
```

The second method allows parameter ordering to be determined at run time. The format string is prepended with a reordering vector specifying the argument order. The reordering vector is a series of bytes with the following format:

```
LDH!<n><o1><o2><o3>...<format string> <n><o1><o2><o3>
etc. are binary bytes.
```

Where:

<n> is the number of arguments in the printf statements. <o1> specifies which argument should be printed first. If the third argument should be printed first, then <o1> would contain the binary value 3. <o2> specifies which argument should be printed second, etc..

NCP Calls

None

See Also

NWprintf, NWsprintf, NWfprintf, NWvsprintf, NWvprintf

NWvprintf

Formats a string and outputs it to standard output. (parameter reordering is supported)

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY(nint) NWvprintf (
   const nstr N_FAR  *format,
   va_list            argList);
```

Parameters

format

(IN) Points to the format string determining how the data will be formatted before sending it to standard output.

argList

(IN) Points to a variable list of parameters whose values are used in the formatted output.

Return Values

Returns the number of bytes output.

Remarks

The variable argument list is given as a single va_list parameter.

For information about formatting data, see the **printf** function in any C manual.

Note that the **NWvprint** function is currently supported on DOS platforms only. This function is the same as the C standard **vprintf** function except that parameter reordering is supported.

The reordering feature allows a language translator to change the order in which arguments are printed by just changing the format string.

There are two ways to do parameter reordering. The most common way is to specify the argument order with a %n in front of each formatting code. For example:

```
NWprintf("There are %d files in the directory %s.\n",
numfiles, dirname);
```

In an internationalized program, the format string would actually be stored in a separate file to be translated into other languages. The following statement changes the order of the arguments with only a change in the format string:

```
NWprintf("Directory %2%s contains %1%d files.\n",
numfiles, dirname);
```

The second method allows parameter ordering to be determined at run time. The format string is prepended with a reordering vector specifying the argument order. The reordering vector is a series of bytes with the following format:

```
LDH!<n><o1><o2><o3>...<format string> <n><o1><o2><o3>
etc. are binary bytes.
```

Where:

<n>   is the number of arguments in the printf statements. <o1> specifies which argument should be printed first. If the third argument should be printed   first, then <o1> would contain the binary value 3. <o2> specifies which argument should be printed second, etc..

NCP Calls

None

See Also

NWprintf, NWsprintf, NWfprintf, NWvfprintf, NWvsprintf

NWvsprintf

Formats a string and outputs it to standard output. (parameter reordering is supported)

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <time.h> or #define NWL_EXCLUDE_TIME
#include <stdio.h> or #define NWL_EXCLUDE_FILE
#include <nwlocale.h>

N_EXTERN_LIBRARY(nint) NWvsprintf (
   pnstr              buffer,
   const nstr N_FAR  *format,
   va_list            argList);
```

Parameters

buffer

(OUT) Points to the buffer receiving the formatted data.

format

(IN) Points to the format string determining how the data will be formatted before sending it to the buffer.

argList

(IN) Points to a variable list of parameters whose values are used in the formatted output.

Return Values

Returns the number of bytes output.

Remarks

The variable argument list is given as a single va_list parameter.

For information about formatting data, see the **printf** function in any C manual.

Note that the **NWvsprintf** function is currently supported on DOS platforms only. This function is the same as the C standard **sprintf** function except that parameter reordering is supported.

The reordering feature allows a language translator to change the order in which arguments are printed by just changing the format string.

There are two ways to do parameter reordering. The most common way is to specify the argument order with a %n in front of each formatting code. For example:

```
NWprintf("There are %d files in the directory %s.\n",
numfiles, dirname);
```

In an internationalized program, the format string would actually be stored in a separate file to be translated into other languages. The following statement changes the order of the arguments with only a change in the format string:

```
NWprintf("Directory %2%s contains %1%d files.\n",
numfiles, dirname);
```

The second method allows parameter ordering to be determined at run time. The format string is prepended with a reordering vector specifying the argument order. The reordering vector is a series of bytes with the following format:

```
LDH!<n><o1><o2><o3>...<format string> <n><o1><o2><o3>
etc. are binary bytes.
```

Where:

<n> is the number of arguments in the printf statements. <o1> specifies which argument should be printed first. If the third argument should be printed first, then <o1> would contain the binary value 3. <o2> specifies which argument should be printed second, etc..

NCP Calls

None

See Also

NWprintf, NWsprintf, NWfprintf, NWvprintf, NWvfprintf

NWwsprintf (obsolete 9/97)

Allows the order of values in the format string to differ from the order of parameter values to be substituted into the format string but is now obsolete. Call **NWsprintf** instead.

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Internationalization

Syntax

```
#include <nwlocale.h>

nint N_API_VARARGS NWwsprintf (
   pnstr   buffer,
   pnstr   format,
          [parameter_1,
           parameter_2]...);
```

Parameters

buffer

(OUT) Points to the buffer receiving the formatted data.

format

(IN) Points to the format string determining how the data will be formatted before sending it to the output buffer.

parameter_1, parameter_2

(IN) Specifies the user-supplied variable list of parameters whose values are used in the formatted output.

Return Values

Returns the number of bytes output.

Remarks

This function contains the same functionality as the **NWsprintf** function, but does not support parameter reordering for all formats on all platforms. This function also differs in that it calls the native OS version of the **vsprintf** function instead of the NetWare supplied version. We recommend that you use **NWsprintf** instead.

NCP Calls

None

RenameLanguage

Changes the name string associated with an OS language ID
Local Servers: nonblocking
Remote Servers: N/A
Classification: 4.x
Platform: NLM
SMP Aware: No
Service: Internationalization

Syntax

```
incl <nwadv.h>

int RenameLanguage (
   int    languageID,
   BYTE  *newLanguageName,
   int    showErrorsToConsole);
```

Parameters

languageID

(IN) Specifies the language ID number for the language to be renamed.

newLanguageName

(IN) Specifies the new name to associate with the language ID.

showErrorsToConsole

(IN) Specifies whether to show error messages to the console screen.

Return Values

0	Success
1	Invalid ID or name

Remarks

RenameLanguage renames the language using the ID specified by the *languageID* parameter.

See Also

AddLanguage

AddLanguage, GetCurrentOSLanguageID, LoadLanguageMessageTable, ReturnLanguageName, SetCurrentOSLanguageID

ReturnLanguageName

Returns the name associated with an OS language ID
Local Servers: nonblocking
Remote Servers: N/A
Classification: 4.x
Platform: NLM
SMP Aware: No
Service: Internationalization

Syntax

```
incl <nwadv.h>

int ReturnLanguageName (
   int    languageID,
   BYTE  *languageName);
```

Parameters

languageID

(IN) Specifies the languageID for which to return the language name.

languageName

(OUT) Receives the language name associated with the specified language ID.

Return Values

0	Successful
1	Invalid language ID

Remarks

The *languageName* parameter receives the name associated with the *languageID* parameter.

See Also

AddLanguage, **GetCurrentOSLanguageID**, **LoadLanguageMessageTable**, **RenameLanguage**, **SetCurrentOSLanguageID**

SetCurrentOSLanguageID

Sets the language ID for the language currently running on the server
Local Servers: nonblocking
Remote Servers: N/A
Classification: 4.x
Platform: NLM
SMP Aware: No
Service: Internationalization

Syntax

```
incl <nwadv.h>

int SetCurrentOSLanguageID (
   LONG   newLanguageID);
```

Parameters

newLanguageID
(IN) Specifies the language ID for the new server language.

Return Values

0	Successful
1	Invalid language ID

Remarks

SetCurrentOSLanguageID changes the language running on the server to the one associated with the *newLanguageID* parameter.

See Also

AddLanguage, **GetCurrentOSLanguageID**, **LoadLanguageMessageTable**, **RenameLanguage**, **ReturnLanguageName**

setlocale

Selects a portion of a program's locale
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
Platform: NLM
SMP Aware: Yes
Service: Internationalization

Syntax

```
#include <locale.h>

char *setlocale (
   int          category,
   const char  *locale);
```

Parameters

category

(IN) Specifies the environment category for the specified locale.

locale

(IN) Specifies the locale for the program.

Return Values

If the selection is successful, a string is returned to indicate the locale that was in effect before **setlocale** was called; otherwise, a NULL pointer is returned.

Remarks

setlocale selects a portion of a locale for a program according to the category given by the *category* parameter and the locale specified by the *locale* parameter.

A locale affects the collating sequence (the order in which characters compare with one another), the way in which certain character-handling functions operate, the decimal-point character that is used in formatted input/output and string conversion, and the format and names used in the time string produced by the **strftime** function.

The possible values for the argument category are as follows:

LC_ALL	Select entire environment
LC_COLL ATE	Select collating sequence
LC_CTYPE	Select the character handling
LC_NUME RIC	Select the numeric-format environment
LC_TIME	Select the time-related environment

At the start of a program, the equivalent of

```
setlocale (LC_ALL, "");
```

is executed.

See Also

localeconv, **NWLsetlocale**, **strftime**

Internationalization: Structures

lconv

Holds locale information for NLM specific functions

Service: Internationalization

Defined In: nwlocale.h

Structure

```
typedef struct {
   char   decimal_point[4];
   char   thousands_sep[4];
   char   grouping[4];
   char   int_curr_symbol[8];
   char   currency_symbol[4];
   char   mon_decimal_point[4];
   char   mon_thousands_sep[4];
   char   mon_grouping[8];
   char   positive_sign[4];
   char   negative_sign[4];
   char   int_frac_digits;
   char   frac_digits;
   char   p_cs_precedes;
   char   p_sep_by_space;
   char   p_sign_posn;
   char   n_cs_precedes;
   char   n_sep_by_space;
   char   n_sign_posn;
} lconv;
```

Fields

decimal_point

Specifies the decimal point character used to format non-monetary quantities.

thousands_sep

Specifies the character used to separate groups of digits to the left of the decimal point character in non-monetary quantities.

grouping

Specifies the size of each group of digits in formatted non-monetary quantities.

int_curr_symbol

Specifies the international currency symbol applicable to the locale.

currency_symbol

Specifies the local currency symbol applicable to the locale.

mon_decimal_point

Specifies the character used as a decimal point in monetary quantities.

mon_thousands_sep

Specifies the character used to separate groups of digits to the left of the decimal point character in formatted monetary quantities.

mon_grouping

Specifies the size of each group of digits in formatted monetary quantities.

positive_sign

Specifies the character used to indicate non-negative monetary quantities.

negative_sign

Specifies the character used to indicate negative monetary quantities.

int_frac_digits

Specifies the number of fractional digits to be displayed in internationally formatted monetary quantities.

frac_digits

Specifies the number of fractional digits to be displayed in locally formatted monetary quantities

p_cs_precedes

Specifies whether the currency symbol precedes or follows the value for non-negative formatted monetary quantities:

0	Currency symbol follows
1	Currency symbol precedes

p_sep_by_space

Specifies whether the currency symbol is separated by a space from the value for non-negative formatted monetary quantities:

0	Currency symbol not separated
1	Currency symbol separated

p_sign_posn

Specifies the position of the *positive_sign* field for a formatted monetary quantity.

n_cs_precedes

Specifies whether the currency symbol precedes or follows the value for negative formatted monetary quantities:

0	Currency symbol precedes
1	Currency symbol follows

n_sep_by_space

Specifies whether the currency symbol is separated by a space from the value for negative formatted monetary quantities:

0	Currency symbol is not separated

0 Currency symbol is not separated

1 Currency symbol is separated

n_sign_posn

Specifies the position of the *negative_sign* field for a formatted monetary quantity.

Remarks

The nwlocale.h file redefines lconv as the LCONV structure and adds additional information.

The *int_curr_symbol* structure stores the international currency symbol applicable to the current locale. The first three characters of the *int_curr_symbol* field contain the alphabetical international currency symbol in accordance with *ISO 4217 Codes for the Representation of Currency and Funds*. The fourth character (preceding the NULL terminator) is the character that separates the international currency symbol from the monetary quantity.

The elements of the *grouping* and *mon_grouping* fields are interpreted as follows:

CHAR_MAX	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remainder of the digits.
other	The value is the number of digits that comprise the current group. The next element is examined to determine the size of the nest group of digits to the left of the current group.

The values of the *p_sign_posn* and *n_sign_posn* fields are interpreted as follows:

0	Parentheses surround the quantity and currency symbol.
1	The sign string precedes the quantity and currency symbol.
2	The sign string follows the quantity and currency symbol.
3	The sign string immediately precedes the quantity and currency symbol.
4	The sign string immediately follows the quantity and currency symbol.

LCONV

Is the NetWare counterpart to the standard lconv structure
Service: Internationalization
Defined In: nwlocale.h

Structure

```
typedef struct {
   char   decimal_point[4];
   char   thousands_sep[4];
   char   grouping[4];
   char   int_curr_symbol[8];
   char   currency_symbol[4];
   char   mon_decimal_point[4];
   char   mon_thousands_sep[4];
   char   mon_grouping[8];
   char   positive_sign[4];
   char   negative_sign[4];
   char   int_frac_digits;
   char   frac_digits;
   char   p_cs_precedes;
   char   p_sep_by_space;
   char   n_cs_precedes;
   char   n_sep_by_space;
   char   p_sign_posn;
   char   n_sign_posn;
   int    code_page;
   int    country_id;
   char   data_list_separator;
   char   date_separator[2];
   char   time_separator[2];
   char   time_format;
   int    date_format;
   char   am[meridlen];
   char   pm[meridlen];
   char   reserved[40];

} LCONV;
```

Pascal Structure

```
Defined in nwlocale.inc

   pLCONV = ^LCONV;
   LCONV = Record
      decimal_point : Array [0..3] of Char;
      thousands_sep : Array [0..3] of Char;
      grouping : Array [0..3] of Char;
```

```
    int_curr_symbol : Array [0..7] of Char;
    currency_symbol : Array [0..3] of Char;
    mon_decimal_point : Array [0..3] of Char;
    mon_thousands_sep : Array [0..3] of Char;
    mon_grouping : Array [0..7] of Char;
    positive_sign : Array [0..3] of Char;
    negative_sign : Array [0..3] of Char;
    int_frac_digits : Char;
    frac_digits : Char;
    p_cs_precedes : Char;
    p_sep_by_space : Char;
    n_cs_precedes : Char;
    n_sep_by_space : Char;
    p_sign_posn : Char;
    n_sign_posn : Char;
    code_page : nint;
    country_id : nint;
    data_list_separator : Array [0..1] of Char;
    date_separator : Array [0..1] of Char;
    time_separator : Array [0..1] of Char;
    time_format : Char;
    date_format : Char;
    am : Array [0..MERIDLEN-1] of Char;
    pm : Array [0..MERIDLEN-1] of Char;
    reserved : Array [0..39] of Char;
End;
```

Fields

decimal_point

Specifies the non-monetary decimal point.

thousands_sep

Specifies the non-monetary separator for digits left of the decimal point.

grouping

Specifies a string showing the size of groups of digits.

int_curr_symbol

Specifies the international currency symbol for the current locale.

currency_symbol

Specifies the currency symbol for the current locale.

mon_decimal_point

Specifies the monetary decimal point.

mon_thousands_sep

Specifies the monetary separator for digits left of the decimal point.

mon_grouping

Specifies a string showing the size of groups of digits.

positive_sign

Specifies a string showing the positive monetary value.

negative_sign

Specifies a string showing the negative monetary value.

int_frac_digits

Specifies the number of fractional digits in monetary display.

frac_digits

Specifies the number of fractional digits in non-monetary display.

p_cs_precedes

Specifies the position of the currency symbol: 1=precede, 0=succeed

p_sep_by_space

Specifies whether to use a space separator with the currency symbol:

0	no space separator
1	space separator

n_cs_precedes

Specifies the location of the currency symbol for a negative monetary quantity.

n_sep_by_space

Specifies the separation of the currency symbol with a negative monetary quantity.

p_sign_posn

Specifies the value showing the position of the positive sign with a positive monetary quantity.

n_sign_posn

Specifies the value showing the position of the negative sign with a negative monetary quantity.

code_page

Specifies the code page of the local system as obtained by the **NWLlocaleconv** function.

country_id

Specifies the country code of the local system.

data_list_separator

Specifies the character used to separate items in a data list.

date_separator

Specifies the character used to separate month, day, and year fields in a date string.

time_separator

Specifies the character used to separate hours, minutes, and seconds in a time string.

time_format

Specifies an 8-bit code indicating the time format to be used.

date_format

Specifies an integer code indicating the date format to be used.

am (Client only)

Specifies a character string containing the local representation for ante meridian (before noon).

pm (Client only)

Specifies a character string containing the local representation for post meridian (after noon).

reserved

Is reserved.

Remarks

The first three characters of the *grouping* field contain the alphabetic international currency symbol in accordance with those specified in ISO 4217. The fourth character is the character used to separate the international currency symbol from the monetary quantity.

The first three characters of the *int_curr_symbol* field contain the alphabetic international currency symbol as defined in ISO 4217, "Codes for the Representation of Currency & Funds."

The *date_format* field has the following format:

```
char   am[MERIDLEN]
char   pm[MERIDLEN]
```

Unicode

Unicode Table API: Guides

Unicode Unicode Table API Concepts

The Unicode Table API provides functions for managing Unicode tables and conversions and for performing operations on Unicode strings.

Introduction to Unicode

Unicode Tables

Unicode Operations

Unicode Table Search Precedence

Types of Unicode Table API Functions

Parent Topic:

Internationalization: Guides

Types of Unicode Table API Functions

The following are the different types of Unicode functions:

Unicode Table Functions

Unicode Buffer and Character Functions

Unicode String Functions

Parent Topic:

Unicode Table API: Guides

Unicode Table API: Examples

Unicode Operations: Example

The following code uses Internationalization services to find a country ID and code page. The example illustrates how Directory Services operations are wedged between the two functions to initialize and free the Unicode tables.

Find Country ID and Code Page

```
/* *******************************************************************
 *
 * Name        :  Initializing the Unicode tables
 *
 *
 * Abstract    : Demonstrate what an application must do to initialize
 *                 unicode tables before using Directory Services.
 *
 * Notes       :
* ********************************************************************
 */


#include <stdio.h>
#include <time.h>
#include <nwlocale.h>
#include <nwdsdc.h>
#include <stdlib.h>
#include <string.h>

void main(void)
{

     NWDSCCODE   ccode;
     char NWFAR  *countryPtr;
     LCONV       lconvInfo;

     countryPtr = NWLsetlocale(LC_ALL, NULL);

/*   Read values from current locale */
     NWLlocaleconv(&lconvInfo);
     ccode = NWInitUnicodeTables(lconvInfo.country_id, lconvInfo.code_p
     printf("\nReturn code from NWInitUnicodeTables is %X\n", ccode);
/*
     Call directory service functions here.
*/

     NWFreeUnicodeTables( );
}
```

Parent Topic:

Unicode Operations

Unicode Table API: Concepts

Unicode Buffer and Character Functions

These functions perform conversion and comparison operations.

Function	Header	Comment
NWUnicodeCompare	unicode.h	Compares a pair of unicode characters.
NWLocalToUnicode	unicode.h	Converts data in the local code page to Unicode.
NWUnicodeToLocal	unicode.h	Converts Unicode data to the local code page.
NWUnicodeToCollation	unicode.h	Converts Unicode to collation.
NWUnicodeToMonocase	unicode.h	Converts Unicode to monocase.

Parent Topic:

Types of Unicode Table API Functions

Unicode Operations

Unicode operations fall into two groups: those specific to managing Unicode tables and conversions, and those that perform operations on Unicode strings. This latter group is the Unicode equivalent of standard string.h functions.

A client application must load Unicode tables for the client agent to perform the conversions. Loading the Unicode tables is typically one of the first steps an application takes in accessing Directory Services. Two functions load and unload the tables:

NWInitUnicodeTables

NWFreeUnicodeTables

As input, **NWInitUnicodeTables** requires a specific code page and country code. You can use Internationalization services to read these parameters from the locale. When you no longer need the tables, call **NWFreeUnicodeTables**.

For many applications, **NWInitUnicodeTables** and **NWFreeUnicodeTables** are the only Unicode functions needed. Other Unicode table requests include the following:

- Return a handle to any of the Unicode table files
- Load and unload one of the Unicode tables
- Compare Unicode characters
- Convert between Unicode and the local code page
- Collate or monocase a Unicode buffer

For more details about specific functions, see Types of Unicode Table API Functions or cousult the specific function reference.

Parent Topic:

Unicode Table API: Guides

Introduction to Unicode

Directory Services stores data in Unicode format, a wide-character encoding scheme that provides the basis for internationalization of information. A Unicode character is 16 bits wide, allowing Directory Services to accommodate the larger character sets used to represent some languages. All character strings exchanged between a directory server and a workstation are in Unicode.

The directory client agent software handles the Unicode format according to the value of the DCV_XLATE_STRINGS flag stored in the directory context associated with the request. If this flag is set, the client agent automatically converts character strings back and forth between Unicode and the local code page so that the client application needs no awareness of Unicode. This is the default setting.

Once the flag is cleared, the client agent no longer performs the conversions, and the client application receives and must submit character strings in Unicode.

The size of buffers required to handle directory results varies depending on whether the client agent is converting Unicode to a local code page. Unicode buffers should be measured in bytes, not characters.

Parent Topic:

Unicode Table API: Guides

Unicode String Functions

These functions operate on Unicode character strings. They provide the functional equivalent of the string.h functions.

Functio n	Header	Comment
unicat	unicode. h	Corresponds to the standard **strcat**.
unichr	unicode. h	Corresponds to the standard **strchr**.
unicpy	unicode. h	Corresponds to the standard **strcpy**.
unicspn	unicode. h	Corresponds to the standard **strcspn**.
uniicm p	unicode. h	Compares two strings for differences.
unilen	unicode. h	Corresponds to the standard **strlen**.
unincat	unicode. h	Corresponds to the standard **strncat**
unincpy	unicode. h	Corresponds to the standard **strncpy**.
uninicm p	unicode. h	Corresponds to the standard **strnicmp**.
uninset	unicode. h	Corresponds to the standard **strnset**.
unipbrk	unicode. h	Corresponds to the standard **strpbrk**.
unipcpy	unicode. h	Corresponds to the standard **strpcpy**.
unirchr	unicode. h	Corresponds to the standard **strrchr**.
unirev	unicode. h	Corresponds to the standard **strrev**.
uniset	unicode. h	Corresponds to the standard **strset**.
unisize	unicode. h	Corresponds to **sizeof**.
unispn	unicode. h	Corresponds to the standard **strspn**.
unistr	unicode.	Corresponds to the standard **strstr**.

	h	
unitok	unicode.h	Corresponds to the standard **strtok**.

Parent Topic:

Types of Unicode Table API Functions

Unicode Table Functions

These functions manipulate Unicode tables.

Function	Header	Comment
NWInitUnicodeTables	unicode.h	Initializes Unicode tables.
NWLoadRuleTable	unicode.h	Loads the specified Unicode table.
NWFreeUnicodeTables	unicode.h	Frees Unicode tables after they have been initialized with **NWInitUnicodeTables**.
NWGetUnicodeToLocalHandle	unicode.h	Returns a handle to the current Unicode to Code Page conversion table.
NWGetLocalToUnicodeHandle	unicode.h	Returns a handle to the current Code Page to Unicode conversion table.
NWGetMonocaseHandle	unicode.h	Returns a handle to the current Unicode Monocasing table.
NWGetCollationHandle	unicode.h	Returns a handle to the current Unicode Collation table.
NWUnloadRuleTable	unicode.h	Deallocates memory for a rule table set up and allocated by NWLoadRuleTable.

Parent Topic:

Types of Unicode Table API Functions

Unicode Table Search Precedence

NWInitUnicodeTables searches for the tables in the following directories in the order they are listed.

1. The current working directory.
2. The directory the application was loaded from.
3. A directory named \nls immediately subordinate to the directory the application was loaded from. (An nls directory is created by the NetWare installation process under sys:system.)
4. A directory named \nls (which is a sibling of the directory from which the application was loaded).
5. A directory in the local search path.

Note that the search path is the last place searched. Consequently, storing the tables in a search path could noticeably increase the amount of time it takes for the tables to load.

Parent Topic:

Unicode Table API: Guides

Unicode Tables

At the workstation, conversions between Unicode and the local code page are supported by a set of conversion tables. The Unicode Filenames table shows the conventions for assigning file names to the tables.

Table auto. Unicode Filenames

Filename	Comment
UNI_<CP>.CTY	Unicode to code page conversion table
<CP>_UNI.CTY	Code page to Unicode conversion table
UNI_COL.CTY	Unicode collation table for country
UNI_MON.CTY	Unicode monocasing table for country

The filenames reflect both a code page (CP) and a country code (CTY). The code page can be 3 or 4 digits. The country code is the country's 3-digit code for long distance telephone numbers. For example, to represent the English character set (CP=437) using U.S. conventions (CTY=001) the following files are required:

UNI_437.001

437_UNI.001
UNI_COL.001
UNI_MON.001

The following shows the code pages supported by Novell.

Table auto. Code Page Support

Country or Language	OEM CP	ANSI CP
U.S., parts of South America and parts of Europe	437	1252
Canada, South America, Western Europe, U. S.	850	1252
Slavic	852	1250
Cyrillic	855	1251
Turkish	857	1254
Portutuese	860	1252
Icelandic	861	1252
Hebrew	862	1255
Canadian French	863	1252
Arabic	864	1256
Scandanvia	865	1252
Russia	866	1251
Greek	869	1253
Thai	874	874
Japan	897	
Japan	932	932
PRC	936	936
Korea	949	949
Taiwan, Hong Kong	950	950

Parent Topic:

Unicode Table API: Guides

Unicode: Functions

NWFreeUnicodeTables

Frees up the memory used by the four Unicode* rule tables created when **NWInitUnicodeTables** was called

NetWare Server: 4.x

Platform: DOS, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWFreeUnicodeTables (
   void);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWFreeUnicodeTables
  : nint;
```

Return Values

0x00 00	Successful is always returned

NCP Calls

None

See Also

NWInitUnicodeTables

NWGetCollationHandle

Sets a handle to the Unicode-to-collation rule table
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.inc>

N_EXTERN_LIBRARY(nint) NWGetCollationHandle (
   pnptr   handle);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWGetCollationHandle
  (handle : pnptr
) : nint;
```

Parameters

handle

(OUT) Points to the rule handle.

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 10	UNI_HANDLE_BAD

Remarks

If the rule table has not been loaded, the pointer is set to NULL.

NCP Calls

None

See Also

NWInitUnicodeTables, **NWLoadRuleTable**, **NWUnicodeToCollation**, **NWUnicodeCompare**

NWGetLocalToUnicodeHandle

Sets rule handle to the local-to-Unicode rule table
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWGetLocalToUnicodeHandle (
   pnptr   handle);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWGetLocalToUnicodeHandle
  (handle : pnptr
) : nint;
```

Parameters

handle
(OUT) Points to the rule handle.

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 10	UNI_HANDLE_BAD

Remarks

If the rule table has not been loaded, the pointer is set to NULL.

NCP Calls

None

See Also

NWInitUnicodeTables, **NWLoadRuleTable**, **NWLocalToUnicode**

NWGetMonocaseHandle

Sets a rule handle to the Unicode-to-monocase rule table
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWGetMonocaseHandle (
   pnptr   handle);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWGetMonocaseHandle
  (handle : pnptr
) : nint;
```

Parameters

handle
(OUT) Points to the rule handle.

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 10	UNI_HANDLE_BAD

Remarks

If the rule table has not been loaded, the pointer is set to NULL.

NCP Calls

None

See Also

NWInitUnicodeTables, **NWLoadRuleTable**, **NWUnicodeToMonocase**

NWGetUnicodeToLocalHandle

Sets a rule handle to the Unicode-to-local rule table
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWGetUnicodeToLocalHandle (
   pnptr   handle);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWGetUnicodeToLocalHandle
  (handle : pnptr
) : nint;
```

Parameters

handle
(OUT) Points to the rule handle.

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 10	UNI_HANDLE_BAD

Remarks

If the rule table has not been loaded, the pointer is set to NULL.

NCP Calls

None

See Also

NWInitUnicodeTables, **NWLoadRuleTable**, **NWUnicodeToLocal**

NWInitUnicodeTables

Sets up and initializes four conversion tables necessary to support Unicode
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWInitUnicodeTables (
   nint   countryCode,
   nint   codePage);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWInitUnicodeTables
  (countryCode : nint;
   codePage    : nint
) : nint;
```

Parameters

countryCode

(IN) Specifies the country or nation identification number.

codePage

(IN) Specifies the Character Encoding Scheme (CES) ID number.

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 11	UNI_LOAD_FAILED
0xFE 12	UNI_NO_MEMORY
0xFE 13	UNI_NO_PERMISSION

0xFE 14	UNI_TOO_MANY_FILES
0xFE 15	UNI_NO_SUCH_FILE
0xFE 16	UNI_FUTURE_OPCODE
0xFE 0B	UNI_OPEN_FAILED
0xFE 0E	UNI_RULES_CORRUPT

Remarks

countryCode and *codePage* can be found by calling **NWLlocaleconv**.

NWInitUnicodeTables or **NWLoadRuleTable** must be successfully called before these calls can be made:

NWGetUnicodeToLocalHandle
NWGetLocalToUnicodeHandle
NWGetMonocaseHandle
NWGetCollationHandle

NCP Calls

None

See Also

NWFreeUnicodeTables, **NWLlocaleconv**, **NWLoadRuleTable**

NWLoadRuleTable

Loads a single rule table when all four conversion or rule tables are not needed

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint)  NWLoadRuleTable (
   pnstr   ruleTableName,
   pnptr   ruleHandle);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWLoadRuleTable
   (ruleTableName : PChar;
    ruleHandle    : pnptr
) : nint;
```

Parameters

ruleTableName

(IN) Points to the full path of the rule table.

ruleHandle

(OUT) Points to the handle to the loaded rule table.

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 11	UNI_LOAD_FAILED
0xFE 12	UNI_NO_MEMORY
0xFE	UNI_NO_PERMISSION

13	
0xFE 14	UNI_TOO_MANY_FILES
0xFE 15	UNI_NO_SUCH_FILE
0xFE 16	UNI_FUTURE_OPCODE
0xFE 0B	UNI_OPEN_FAILED
0xFE 0E	UNI_RULES_CORRUPT

Remarks

NWLoadRuleTable replaces **NWInitUnicodeTables** and the following calls:

NWGetUnicodeToLocalHandle
NWGetLocalToUnicodeHandle
NWGetMonocaseHandle
NWGetCollationHandle

For information about rule tables, see **NWInitUnicodeTables**.

NCP Calls

None

See Also

NWInitUnicodeTables, **NWUnloadRuleTable**

NWLocalToUnicode

Converts a local (code page based) character string to a Unicode character string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWLocalToUnicode (
   nptr        ruleHandle,
   punicode    dest,
   nuint32     maxLen,
   nptr        src,
   unicode     noMap,
   pnuint      len);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWLocalToUnicode
  (ruleHandle     : nptr;
   dest           : punicode;
   maxLen         : nuint32;
   src            : nptr;
   noMap          : unicode;
   len            : pnuint);
```

Parameters

ruleHandle

(IN) Points to the rule table handle for local to Unicode conversion as initialized by **NWGetLocalToUnicodeHandle**.

dest

(OUT) Points to the buffer for storing the resulting Unicode string.

maxLen

(IN) Specifies the maximum number of Unicode characters in the *dest* parameter (each unicode character is 2 bytes).

src

(IN) Points to the source-local character string.

noMap

(IN) Specifies the no map character.

len

(OUT) Points to the number of characters copied into the*dest* parameter (including the 2-byte null termination character).

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 10	UNI_HANDLE_BAD
0xFE 0D	UNI_NO_DEFAULT
0xFE 0E	UNI_RULES_CORRUPT
0xFE 0F	UNI_HANDLE_MISMATCH

Remarks

On Win32 clients, this routine calls MultiByteToWideChar which results in some differences from other platforms. If there is insufficient space in the output buffer, Win32 clients set len to zero, and do not null terminate the buffer. Non-Win32 clients always return the number of unicode characters written into the output buffer and always null terminate

If an input character is unmappable, Win32 clients use a default substitute character, not the noMap character. On non-Win32 clients, an unmappable character is replaced by the noMap character. If the noMap character is zero, it uses the default character contained in the rule table. If the rule table has no default, UNI_NO_DEFAULT is returned.

NCP Calls

None

See Also

NWGetLocalToUnicodeHandle, **NWUnicodeToLocal**

NWUnicodeCompare

Compares one Unicode character to another after having converted them using the unicode-to-collation rule table

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWUnicodeCompare (
   nptr       ruleHandle,
   unicode    chr1,
   unicode    chr2);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWUnicodeCompare
  (ruleHandle : nptr;
   char1      : unicode;
   char2      : unicode
) : nint;
```

Parameters

ruleHandle

(IN) Points to the rule table handle for unicode-to-collation conversion.

chr1

(IN) Specifies the 1st character.

chr2

(IN) Specifies the 2nd character.

Return Values

< 0	If chr1 < chr2
= 0	If chr1 = chr2
>	If chr1 > chr2

0

NCP Calls

None

See Also

NWGetCollationHandle, **NWUnicodeToCollation**

NWUnicodeToCollation

Converts a Unicode string to an array of collation weights for use in comparison operations to provide proper sorting for the current country

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWUnicodeToCollation (
   nptr        ruleHandle,
   punicode    dest,
   nuint32     maxLen,
   punicode    src,
   unicode     noMap,
   pnuint32    len);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWUnicodeToCollation
  (ruleHandle :   nptr;
   dest       :   punicode;
   maxLen     :   nuint32;
   src        :   punicode;
   noMap      :   unicode;
   len        :   pnuint32
) : nint;
```

Parameters

ruleHandle

(IN) Points to the rule table handle for unicode-to-collation conversion.

dest

(OUT) Points to the buffer for resulting Unicode collation weights.

maxLen

(IN) Specifies the maximum number of Unicode characters in the *dest* parameter. The buffer is null terminated.

src

(IN) Points to the buffer with source Unicode.

noMap

(IN) Specifies no map character.

len

(OUT) Points to the number of characters copied into the*dest* parameter, including the 2-byte null termination character.

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 10	UNI_HANDLE_BAD
0xFE 0D	UNI_NO_DEFAULT
0xFE 0E	UNI_RULES_CORRUPT
0xFE 0F	UNI_HANDLE_MISMATCH

Remarks

On Win32 clients, this routine calls wcsxfrm which results in some differences from other platforms. If there is insufficient space in the output buffer, Win32 clients set len = maxLen, do not null terminate the output buffer, and return status -1. Non-Win32 clients always return the number of unicode characters written into the output buffer and always null terminate.

NWUnicodeToCollation is similar to the C **strxfrm**.

NCP Calls

None

See Also

NWGetCollationHandle, **NWUnicodeCompare**

NWUnicodeToLocal

Converts a Unicode character string to local (code page based) character string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY (nint) NWUnicodeToLocal (
   nptr            ruleHandle,
   pnuint8         dest,
   size_t          maxLen,
   punicode        src,
   nuint8          noMap,
   size_t N_FAR   *len);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWUnicodeToLocal
  (ruleHandle    : nptr;
   dest          : nptr;
   maxLen        : nuint32;
   src           : punicode;
   noMap         : nuint8;
   len           : pnuint);
```

Parameters

ruleHandle

(IN) Points to the rule table handle for Unicode-to-local conversion.

dest

(OUT) Points to the buffer for resulting character string.

maxLen

(IN) Specifies the maximum number of bytes in *dest*.

src

(IN) Points to the buffer containing the source Unicode.

noMap

(IN) Specifies the no map character.

len

(OUT) Points to the number of characters copied into the *dest* parameter (including the null byte).

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 10	UNI_HANDLE_BAD
0xFE 0D	UNI_NO_DEFAULT
0xFE 0E	UNI_RULES_CORRUPT
0xFE 0F	UNI_HANDLE_MISMATCH

Remarks

On Win32 clients, this routine calls WideCharToMultiByte which results in some differences from other platforms. If there is insufficient space in the output buffer, Win32 clients set len to zero and do not null terminate the buffer. Non-Win32 clients always return the number of bytes written into the output buffer and always null terminate.

If the noMap character is zero and an unmappable character is encountered, Win32 clients use the system default substitution character. Non-Win32 clients use the default character contained in the rule table. If the rule table has no default, then error code UNI_NO_DEFAULT is returned.

NCP Calls

None

See Also

NWGetUnicodeToLocalHandle, **NWLocalToUnicode**

NWUnicodeToMonocase

Converts a mixed upper/lower case Unicode string to a monocase Unicode string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWUnicodeToMonocase (
   nptr        ruleHandle,
   punicode    dest,
   nuint32     maxLen,
   punicode    src,
   pnuint32    len);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWUnicodeToMonocase
  (ruleHandle : nptr;
   dest       : punicode;
   maxLen     : size_t;
   src        : punicode;
   len        : psize_t
) : nint;
```

Parameters

ruleHandle

(IN) Points to the rule table handle for unicode-to-monocase conversion.

dest

(OUT) Points to the buffer for the resulting monocase Unicode string.

maxLen

(IN) Specifies the maximum number of Unicode characters in the *dest* parameter, including the NULL terminator.

src

(IN) Points to the buffer with source Unicode.

len

(OUT) Points to the number of characters copied to the *dest* parameter

(OUT) Points to the number of characters copied to the *dest* parameter including the NULL terminator.

Return Values

These are common return values; see Return Values for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD
0xFE0E	UNI_RULES_CORRUPT
0xFE0F	UNI_HANDLE_MISMATCH

Remarks

NWUnicodeToMonocase converts a Unicode string having a mixture of upper/lower case characters to a Unicode string consistently having only one case, according to the monocase rule table.

For Windows 95, **NWUnicodeToMonocase** is limited to a 256-character string.

NCP Calls

None

See Also

NWGetMonocaseHandle

NWUnloadRuleTable

Deallocates memory for a rule table set up and allocated by **NWLoadRuleTable**

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWUnloadRuleTable (
   nptr   ruleHandle);
```

Pascal Syntax

```
#include <unicode.inc>

Function NWUnloadRuleTable
  (ruleHandle : nptr
) : nint;
```

Parameters

ruleHandle

(IN) Points to the rule table handle.

Return Values

These are common return values; see Return Values for more information.

0x00 00	Successful
0xFE 10	UNI_HANDLE_BAD

NCP Calls

None

See Also

NWLoadRuleTable

NWUSByteToUnicode

Converts a NULL-terminated byte string into a Unicode string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSByteToUnicode (
   punicode   unicodeOutput,
   nuint      outputBufferLen,
   pnuint8    byteInput,
   pnuint     actualLength);
```

Parameters

unicodeOutput

(OUT) Points to the output buffer to receive the resulting Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters.

byteInput

(IN) Points to the input buffer containing the byte text to be converted.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	NWU_NO_CONVERTER

Remarks

Call **NWUSByteToUnicodePath** whenever the byte string to be converted is a file path.

NWUSByteToUnicode converts an unmappable byte into the substitute Unicode character.

NWUSByteToUnicode converts a special byte sequence into a Unicode character. For example, "[2620]" is converted to the single Unicode character, 0x2620.

Call **NWUSByteToUnicode** to determine the size of the string before it is converted by setting the *unicodeOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUSByteToUnicode: Example.

NCP Calls

None

See Also

NWUSByteToUnicodePath

NWUSByteToUnicodePath

Converts a NULL-terminated file path byte string into a Unicode string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSByteToUnicodePath (
   punicode    unicodeOutput,
   nuint       outputBufferLen,
   pnuint8     byteInput,
   pnuint      actualLength);
```

Parameters

unicodeOutput

(OUT) Points to the output buffer to receive the resulting Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters.

byteInput

(IN) Points to the input buffer containing the byte string to be converted.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	NWU_NO_CONVERTER

Remarks

Call **NWUSByteToUnicodePath** whenever the byte string to be converted is a file path.

NWUSByteToUnicodePath converts an unmappable byte into the substitute Unicode character.

NWUSByteToUnicodePath converts a special byte sequence into a Unicode character. For example, "[2620]" is converted to the single Unicode character, 0x2620.

Call **NWUSByteToUnicodePath** to determine the size of the string before it is converted by setting the *unicodeOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUSByteToUnicodePath: Example.

NCP Calls

None

See Also

NWUSByteToUnicode

NWUSGetCodePage

Returns the codepage used to specify which converters are loaded
Local Servers: to be supplied
Remote Servers: to be supplied
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSGetCodePage (
   pnuint   pCodePage,
   pnuint   pCountry);
```

Parameters

pCodePage
(OUT) Points to the local code page used for the Unicode/byte converter.

pCountry
(OUT) Points to the local country code.

Return Values

NWUSGetCodePage always returns zero.

Remarks

NWUSGetCodePage does not require Unicode initialization and might be useful for printing an error message to tell the user which converters are needed.

DOS or a DOS window returns the local OEM code page. GUI applications return the local ANSI code page.

See NWUSGetCodePage: Example.

NCP Calls

None

NWUSLenByteToUnicode

Converts a length-specified byte string into a Unicode string
Local Servers: to be supplied
Remote Servers: to be supplied
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSByteToUnicode (
   punicode   unicodeOutput,
   nuint      outputBufferLen,
   pnuint8    byteInput,
   nuint      inLength,
   pnuint     actualLength);
```

Parameters

unicodeOutput

(OUT) Points to the output buffer to receive the resulting Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters.

byteInput

(IN) Points to the input buffer containing the byte text to be converted.

inLength

(IN) Specifies the length of the input string in bytes (might not be NULL-terminated).

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD	NWU_BUFFER_FULL

EE	
0xFD E0	NWU_NO_CONVERTER

Remarks

If NULL is encountered in the input string, it is treated as the end of the string and the remaining characters are ignored.

Call **NWUSLenByteToUnicodePath** whenever the byte string to be converted is a file path.

NWUSLenByteToUnicode converts an unmappable byte into the substitute Unicode character.

NWUSLenByteToUnicode converts a special byte sequence into a Unicode character. For example, "[2620]" is converted to the single Unicode character, 0x2620.

Call **NWUSLenByteToUnicode** to determine the size of the string before it is converted by setting the *unicodeOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

To determine the behavior of **NWUSLenByteToUnicode** when an embedded NULL is encountered, seeLength-Specified Byte String Conversion.

See NWUSLenByteToUnicode: Example.

NCP Calls

None

See Also

NWUSLenByteToUnicodePath, **NWUSUnicodeToUntermByte**

NWUSLenByteToUnicodePath

Converts a length-specified file path byte string into a Unicode string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSLenByteToUnicodePath (
   punicode   unicodeOutput,
   nuint      outputBufferLen,
   pnuint8    byteInput,
   nuint      inLength,
   pnuint     actualLength);
```

Parameters

unicodeOutput

(OUT) Points to the output buffer to receive the resulting Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters.

byteInput

(IN) Points to the input buffer containing the byte string to be converted.

inLength

(IN) Specifies the length of the input string in bytes (might not be NULL-terminated).

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL

0xFD EE	NWU_BUFFER_FULL
0xFD E0	NWU_NO_CONVERTER

Remarks

Call **NWUSLenByteToUnicodePath** whenever the byte string to be converted is a file path.

NWUSLenByteToUnicodePath converts an unmappable byte into the substitute Unicode character.

NWUSLenByteToUnicodePath converts a special byte sequence into a Unicode character. For example, "[2620]" is converted to the single Unicode character, 0x2620.

Call **NWUSLenByteToUnicodePath** to determine the size of the string before it is converted by setting the *unicodeOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

To determine the behavior of **NWUSLenByteToUnicodePath** when an embedded NULL is encountered, seeLength-Specified Byte String Conversion.

See NWUSLenByteToUnicodePath: Example.

NCP Calls

None

See Also

NWUSLenByteToUnicode, **NWUSUnicodeToUntermBytePath**

NWUSStandardUnicodeInit

Loads both the lower and upper case converters needed for the standard Unicode functions

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSStandardUnicodeInit (
   void);
```

Return Values

0x00 00	SUCCESSFUL
0xFD E8	NWU_CONVERTER_CORRUPT
0xFD E6	NWU_OUT_OF_MEMORY
0xFD E5	NWU_READ_FAILED (DOS only)
0xFD E4	NWU_OPEN_FAILED (DOS only)
0xFD E3	NWU_NO_PERMISSION (DOS only)
0xFD E2	NWU_TOO_MANY_FILES (DOS only)
0xFD E1	NWU_CONVERTER_NOT_FOUND

Remarks

The **NWUSStandardUnicodeInit** function is equivalent to:

loading the appropriate Unicode to byte converter for the current code page loading a lower case and an upper case converter

setting the converter to use the default substitution character for unmappable bytes

enabling the default no map and scan/parse Unicode handlers

NWUSStandardUnicodeInit can be nested. However, each time **NWUSStandardUnicodeInit** is called, the **NWUSStandardUnicodeRelease** function must be called to release associated resources.

On the NLM platform, the standard converter is initialized when the LOCNLM32.NLM file is loaded. It is still recommended that your applications call **NWUSStandardUnicodeInit** and the **NWUSStandardUnicodeRelease** function explicitly.

See NWUSStandardUnicodeInit: Example.

NCP Calls

None

See Also

NWUSStandardUnicodeRelease

NWUSStandardUnicodeRelease

Releases all resources allocated by the **NWUSStandardUnicodeInit** function

Local Servers: nonblocking

Remote Servers: nonblocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(void) NWUSStandardUnicodeRelease (
   void);
```

Return Values

None

Remarks

Call the **NWUSStandardUnicodeRelease** function only after calling the **NWUSStandardUnicodeInit** function. Otherwise, resources required by another application could be released

See NWUSStandardUnicodeRelease: Example.

NCP Calls

None

See Also

NWUSStandardUnicodeInit

NWUSUnicodeToByte

Converts a NULL-terminated Unicode string into a byte string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToByte (
   pnuint8     byteOutput,
   nuint       outputBufferLen,
   punicode    unicodeInput,
   pnuint      actualLength);
```

Parameters

byteOutput

(OUT) Points to the output buffer to receive the resulting byte string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes.

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in bytes) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD EE	NWU_BUFFER_FULL
0xFD E0	NWU_NO_CONVERTER

Remarks

Call **NWUSUnicodeToBytePath** whenever the Unicode string to be converted is a file path.

The NWU_BUFFER_FULL error can occur because the output buffer has been allocated with insufficient space. Because of the default conversion behavior described above, byte strings can be as much as six times longer than the input string.

NWUSUnicodeToByte converts an unmappable Unicode character into a special 6-byte sequence. For example, 0x2620 is converted to "[2620]".

NWUSUnicodeToByte does not recognize any special Unicode sequences for conversion. For example, the Unicode string "ab[81]cd" will be returned as the byte string "ab[81]cd".

Call **NWUSUnicodeToByte** to determine the size of the string before it is converted by setting the *byteOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUSUnicodeToByte: Example.

NCP Calls

None

See Also

NWUSUnicodeToBytePath

NWUSUnicodeToBytePath

Converts a NULL-terminated file path Unicode string into a byte string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToBytePath (
   pnuint8     byteOutput,
   nuint       outputBufferLen,
   punicode    unicodeInput,
   pnuint      actualLength);
```

Parameters

byteOutput

(OUT) Points to the output buffer to receive the resulting byte string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes.

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in bytes) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD EE	NWU_BUFFER_FULL
0xFD E0	NWU_NO_CONVERTER

Remarks

Call **NWUSUnicodeToBytePath** whenever the Unicode string to be converted is a file path.

The NWU_BUFFER_FULL error can occur because the output buffer has been allocated with insufficient space. Because of the default conversion behavior described above, byte strings can be as much as six times longer than the input string.

Regardless of the language being converted, **NWUSUnicodeToBytePath** interprets Unicode yen (00A5), won (20A9), backslash (005C), and the Novell defined path separator (F8F7) all as path separators and converts each to the byte backslash character (5C).

NWUSUnicodeToBytePath converts an unmappable Unicode character into a special 6-byte sequence. For example, 0x2620 is converted to "[2620]".

NWUSUnicodeToBytePath does not recognize any special Unicode sequences for conversion. For example, the Unicode string "ab[81]cd" will be returned as the byte string "ab[81]cd".

Call **NWUSUnicodeToBytePath** to determine the size of the string before it is converted by setting the *byteOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUSUnicodeToBytePath: Example.

NCP Calls

None

See Also

NWUSUnicodeToByte

NWUSUnicodeToLowerCase

Converts a NULL-terminated Unicode string to Unicode lower case characters

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToLowerCase (
   punicode   lowerCaseOutput,
   nuint      outputBufferLen,
   punicode   unicodeInput,
   pnuint     actualLength);
```

Parameters

lowerCaseOutput

(OUT) Points to the output buffer to receive the resulting lower case string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters.

unicodeInput

(IN) Points to the input buffer containing the Unicode string to convert.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD EE	NWU_BUFFER_FULL
0xFD E0	NWU_NO_CONVERTER

Remarks

Lower case conversions are preferred for most operations since there are more lower case characters than upper case characters in the Unicode environment.

Call **NWUSUnicodeToLowerCase** to determine the size of the string before it is converted by setting the *lowerCaseOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUSUnicodeToLowerCase: Example.

NCP Calls

None

NWUSUnicodeToUpperCase

Converts a NULL-terminated Unicode string to Unicode upper case characters

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToUpperCase (
   punicode   upperCaseOutput,
   nuint      outputBufferLen,
   punicode   unicodeInput,
   pnuint     actualLength);
```

Parameters

upperCaseOutput

(OUT) Points to the output buffer to receive the resulting upper case string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters.

unicodeInput

(IN) Points to the input buffer containing the Unicode string to convert.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD EE	NWU_BUFFER_FULL
0xFD E0	NWU_NO_CONVERTER

Remarks

Lower case conversions are preferred for most operations since there are more lower case characters than upper case characters in the Unicode environment.

Call **NWUSUnicodeToUpperCase** to determine the size of the string before it is converted by setting the *upperCaseOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUSUnicodeToUpperCase: Example.

NCP Calls

None

NWUSUnicodeToUntermByte

Converts a NULL-terminated Unicode string into an unterminated byte string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToUntermByte (
   pnuint8     byteOutput,
   nuint       outputBufferLen,
   punicode    unicodeInput,
   pnuint      actualLength);
```

Parameters

byteOutput

(OUT) Points to the output buffer to receive the resulting unterminated byte string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes.

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in bytes) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD EE	NWU_BUFFER_FULL
0xFD E0	NWU_NO_CONVERTER

Remarks

The NWU_BUFFER_FULL error can occur because the output buffer has been allocated with insufficient space. Because of the default conversion behavior, byte strings can be as much as six times longer than the input string.

NWUSUnicodeToUntermByte converts an unmappable Unicode character into a special unterminated 6-byte sequence. For example, 0x2620 is converted to "[2620]".

NWUSUnicodeToUntermByte does not recognize any special Unicode sequences for conversion. For example, the Unicode string "ab[81]cd" will be returned as the byte string "ab[81]cd".

Call **NWUSUnicodeToUntermByte** to determine the size of the string before it is converted by setting the *byteOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUSUnicodeToUntermByte: Example.

NCP Calls

None

See Also

NWUSLenByteToUnicode, **NWUSUnicodeToUntermBytePath**

NWUSUnicodeToUntermBytePath

Converts a NULL-terminated file path Unicode string into an unterminated byte string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToUntermBytePath (
   pnuint8     byteOutput,
   nuint       outputBufferLen,
   punicode    unicodeInput,
   pnuint      actualLength);
```

Parameters

byteOutput

(OUT) Points to the output buffer to receive the resulting unterminated byte string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes.

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in bytes) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD EE	NWU_BUFFER_FULL
0xFD E0	NWU_NO_CONVERTER

Remarks

Call **NWUSUnicodeToUntermBytePath** whenever the Unicode string to be converted is a file path.

The NWU_BUFFER_FULL error can occur because the output buffer has been allocated with insufficient space. Because of the default conversion behavior, byte strings can be as much as six times longer than the input string.

Regardless of the language being converted, **NWUSUnicodeToUntermBytePath** interprets Unicode yen (00A5), won (20A9), backslash (005C), and the Novell defined path separator (F8F7) all as path separators and converts each to the byte backslash character (5C).

NWUSUnicodeToUntermBytePath converts an unmappable Unicode character into a special unterminated 6-byte sequence. For example, 0x2620 is converted to "[2620]".

NWUSUnicodeToUntermBytePath does not recognize any special Unicode sequences for conversion. For example, the Unicode string "ab[81]cd" will be returned as the byte string "ab[81]cd".

Call **NWUSUnicodeToUntermBytePath** to determine the size of the string before it is converted by setting the *byteOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUSUnicodeToUntermBytePath: Example.

NCP Calls

None

See Also

NWUSLenByteToUnicodePath, **NWUSUnicodeToUntermByte**

NWUXByteToUnicode

Converts a NULL-terminated byte string into a Unicode string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXByteToUnicode (
   pCONVERT N_FAR  *byteUniHandle,
   punicode         unicodeOutput,
   nuint            outputBufferLen,
   pnuint8          byteInput,
   pnuint           actualLength);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

unicodeOutput

(OUT) Points to the buffer to receive the converted Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters (0 specifies that the buffer size is not checked).

byteInput

(IN) Points to the input buffer containing the byte string to be converted.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD	NWU_NULL_HANDLE

E9	
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EC	NWU_UNMAPPABLE_CHAR
0xFD EE	NWU_BUFFER_FULL

Remarks

Passing a zero to the *outputBufferLen* parameter specifies that the size of the output buffer is not checked. It is the responsibility of the caller to assure a sufficient output buffer size.

If NWU_UNMAPPABLE_CHAR is returned, the buffer is not filled past the unmappable character, but is valid up to that character.

By default, **NWUXByteToUnicode** converts an unmappable byte into the substitute Unicode character.

By default, **NWUXByteToUnicode** converts a special byte sequence into a Unicode character. For example, "[2620]" is converted to the single Unicode character, 0x2620.

Call **NWUXByteToUnicode** to determine the size of the string before it is converted by setting the *unicodeOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXByteToUnicode: Example.

NCP Calls

None

See Also

NWUXByteToUnicodePath

NWUXByteToUnicodePath

Converts a NULL-terminated file path byte string into a Unicode string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXByteToUnicode (
   pCONVERT N_FAR  *byteUniHandle,
   punicode         unicodeOutput,
   nuint            outputBufferLen,
   pnuint8          byteInput,
   pnuint           actualLength);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

unicodeOutput

(OUT) Points to the buffer to receive the converted Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters (0 specifies that the buffer size is not checked).

byteInput

(IN) Points to the input buffer containing the byte file path string to be converted.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD	NWU_NULL_HANDLE

E9	
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EC	NWU_UNMAPPABLE_CHAR
0xFD EE	NWU_BUFFER_FULL

Remarks

When you want to convert a file path, call the **NWUXByteToUnicodePath** function rather than the **NWUXByteToUnicode** function.

Passing a zero to the *outputBufferLen* parameter specifies that the size of the output buffer is not checked. It is the responsibility of the caller to assure a sufficient output buffer size.

If NWU_UNMAPPABLE_CHAR is returned, the buffer is not filled past the unmappable character, but is valid up to that character.

By default, **NWUXByteToUnicodePath** converts an unmappable byte into the substitute Unicode character.

By default, **NWUXByteToUnicodePath** converts a special byte sequence into a Unicode character. For example, "[2620]" is converted to the single Unicode character, 0x2620.

Call **NWUXByteToUnicodePath** to determine the size of the string before it is converted by setting the *unicodeOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXByteToUnicodePath: Example.

NCP Calls

None

See Also

NWUXByteToUnicode

NWUXGetByteFunctions

Returns the functions used for handling unmappable bytes
Local Servers: to be supplied
Remote Servers: to be supplied
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetByteFunctions (
   pCONVERT    byteUniHandle,
   NMBYTE     *noMapByteFunc,
   SCBYTE     *scanByteFunc,
   PRBYTE     *parseByteFunc);
```

Parameters

byteUniHandle
(IN) Points to the converter handle.

noMapByteFunc
(OUT) Points to the function to be called when an unmappable byte sequence is found (or set to NULL).

scanByteFunc
(OUT) Points to the function to scan for special data (or set to NULL).

parseByteFunc
(OUT) Points to the function to parse special data (or set to NULL).

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH

Remarks

The system-supplied NoMap byte function, which is not enabled by default, converts an unmappable byte 0xNN into the unicode string "[NN]".

The system-supplied scan/parse byte functions, which are enabled by default, scan for occurrences of the byte string "[NNNN]" (NNNN being any four hexadecimal digits) and converts the string to the single Unicode character 0xNNNN.

See NWUXGetByteFunctions: Example.

NCP Calls

None

See Also

NWUXGetNoMapAction, **NWUXGetUniFunctions**, **NWUXGetSubByte**, **NWUXGetSubUni**, **NWUXSetNoMapAction**, **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, **NWUXSetSubByte**, **NWUXSetSubUni**

NWUXGetCharSize

Returns the character size (1 or 2) of the next character in the specified byte string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetCharSize (
   pCONVERT   byteUniHandle,
   pnuint8    byteInput,
   pnuint     charSize);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

byteInput

(IN) Points to a single or double-byte character in a byte string.

charSize

(OUT) Points to the returned character size.

Return Values

0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

Remarks

NWUXGetCharSize is provided so that user-written Scan and Parse functions are able to examine a byte string.

NOTE: The **NWCharType** function interprets the specified character according to the native system codepage. **NWUXGetCharSize** uses the codepage of the specified converter.

See NWUXGetCharSize: Example.

NCP Calls

None

NWUXGetNoMapAction

Returns the actions to follow when an unmappable byte sequence and an unmappable Unicode character are found

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetNoMapAction (
   pCONVERT   byteUniHandle,
   pnint      noMapByteAction,
   pnint      noMapUniAction);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

noMapByteAction

(OUT) Points to the action to follow for unmappable bytes (optional, set to NULL if not needed).

noMapUniAction

(OUT) Points to the action to follow for unmappable Unicode characters (optional, set to NULL if not needed).

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

Remarks

The *noMapByteAction* and *noMapUniAction* parameters return one of the following values:

NWU_RETURN_ERROR	
NWU_SUBSTITUTE	(Default for *noMapByteAction*)
NWU_CALL_HANDLER	(Default for *noMapUniAction*)

See NWUXGetNoMapAction: Example.

NCP Calls

None

See Also

NWUXGetByteFunctions, **NWUXGetUniFunctions**, **NWUXGetSubByte**, **NWUXGetSubUni**, **NWUXSetNoMapAction**, **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, **NWUXSetSubByte**, **NWUXSetSubUni**

NWUXGetScanAction

Gets the status of current scan/parse functions
Local Servers: nonblocking
Remote Servers: nonblocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetScanAction (
   pCONVERT   byteUniHandle,
   pnint      scanByteAction,
   pnint      scanUniAction);
```

Parameters

byteUniHandle
(IN) Points to the converter handle.

scanByteAction
(OUT) Points to the status of current byte scan/parse functions (optional, set to NULL if not needed).

scanUniAction
(OUT) Points to the status for current Unicode scan/parse functions (optional, set to NULL if not needed).

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD ED	NWU_RANGE_ERROR

Remarks

The *scanByteAction* and *scanUniAction* parameters return one of the following values:

NWU_ENABLED	Scan/parse functions are enanbled
NWU_DISABLED	Scan/parse functions are disabled

The default state of the *scanByteAction* parameter is NWU_DISABLED after a converter is initialized.

The default state of the *scanUniAction* parameter is NWU_ENABLED.

See NWUXGetScanAction: Example.

NCP Calls

None

See Also

NWUXSetScanAction, **NWUXGetByteFunctions**, **NWUXGetUniFunctions**

NWUXGetSubByte

Returns the substitution byte for the specified converter

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetSubByte (
   pCONVERT   byteUniHandle,
   pnuint8    substituteByte);
```

Parameters

byteUniHandle

(IN) Points to the handle of the converter to be used.

substituteByte

(OUT) Points to the current substitution byte (see Substitution Characters).

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

See NWUXGetSubByte: Example.

NCP Calls

None

See Also

NWUXGetNoMapAction, **NWUXGetByteFunctions**, **NWUXGetUniFunctions**, **NWUXGetSubUni**, **NWUXSetNoMapAction**, **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, **NWUXSetSubByte**, **NWUXSetSubUni**

NWUXGetSubUni

Returns the current substitution Unicode character for the converter

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUGetSubUni (
   pCONVERT   byteUniHandle,
   punicode   substituteUni);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

substituteUni

(OUT) Points to the current substitution character (see Substitution Characters).

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

See NWUXGetSubUni: Example.

NCP Calls

None

See Also

NWUXGetNoMapAction, **NWUXGetByteFunctions**, **NWUXGetUniFunctions**, **NWUXGetSubByte**, **NWUXSetNoMapAction**, **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, **NWUXSetSubByte**, **NWUXSetSubUni**

NWUXGetUniFunctions

Returns the functions used for handling unmappable Unicode characters
Local Servers: to be supplied
Remote Servers: to be supplied
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetUniFunctions (
   pCONVERT    byteUniHandle,
   NMUNI      *noMapUniFunc,
   SCUNI      *scanUniFunc,
   PRUNI      *parseUniFunc);
```

Parameters

byteUniHandle
(IN) Points to the converter handle.

noMapByteFunc
(OUT) Points to the Unicode function to be called when unmappable Unicode characters are found.

scanByteFunc
(OUT) Points to the Unicode function to scan for special data.

parseByteFunc
(OUT) Points to the Unicode function to parse special data.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

Remarks

The system-supplied NoMap Unicode function, which is enabled by default, converts an unmappable Unicode character 0xNNNN into the byte string "[NNNN]".

The system-supplied scan/parse Unicode functions, which are not enabled by default, scan for occurrences of the Unicode string "[NN]" (NN being any two hexadecimal digits) and converts the string to the single byte character 0xNN.

See NWUXGetUniFunctions: Example.

NCP Calls

None

See Also

NWUXGetNoMapAction, **NWUXGetByteFunctions**, **NWUXGetSubByte**, **NWUXGetSubUni**, **NWUXSetNoMapAction**, **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, **NWUXSetSubByte**, **NWUXSetSubUni**

NWUXLenByteToUnicode

Converts a length-specified byte string into a Unicode string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLenByteToUnicode (
   pCONVERT   byteUniHandle,
   punicode   unicodeOutput,
   nuint      outputBufferLen,
   pnuint8    byteInput,
   nuint      inLength,
   pnuint     actualLength);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

unicodeOutput

(OUT) Points to the buffer to receive the converted Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters (0 specifies that the buffer size is not checked).

byteInput

(IN) Points to the input buffer containing the byte string to be converted.

inLength

(IN) Specifies the length of the input string in bytes (might not be NULL-terminated).

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string not including the NULL terminator (optional).

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EC	NWU_UNMAPPABLE_CHAR
0xFD EE	NWU_BUFFER_FULL

Remarks

Passing a zero to the *outputBufferLen* parameter specifies that the size of the output buffer is not checked. It is the responsibility of the caller to assure a sufficient output buffer size.

If NWU_UNMAPPABLE_CHAR is returned, the buffer is not filled past the unmappable character, but is valid up to that character.

By default, **NWUXLenByteToUnicode** converts an unmappable byte into the substitute Unicode character.

By default, **NWUXLenByteToUnicode** converts a special byte sequence into a Unicode character. For example, "[2620]" is converted to the single Unicode character, 0x2620.

Call **NWUXLenByteToUnicode** to determine the size of the string before it is converted by setting the *unicodeOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

To determine the behavior of **NWUXLenByteToUnicode** when an embedded NULL is encountered, seeLength-Specified Byte String Conversion.

See NWUXLenByteToUnicode: Example.

NCP Calls

None

See Also

NWUXLenByteToUnicodePath, NWUXUnicodeToUntermByte

NWUXLenByteToUnicodePath

Converts a length-specified file path byte string into a Unicode string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLenByteToUnicode (
   pCONVERT   byteUniHandle,
   punicode   unicodeOutput,
   nuint      outputBufferLen,
   pnuint8    byteInput,
   nuint      inLength,
   pnuint     actualLength);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

unicodeOutput

(OUT) Points to the buffer to receive the converted Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in Unicode characters (0 specifies that the buffer size is not checked).

byteInput

(IN) Points to the input buffer containing the byte file path string to be converted.

inLength

(IN) Specifies the length of the input string in bytes (might not be NULL-terminated).

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EC	NWU_UNMAPPABLE_CHAR
0xFD EE	NWU_BUFFER_FULL

Remarks

When you want to convert a file path, call **NWUXLenByteToUnicodePath** rather than the **NWUXLenByteToUnicode** function.

Passing a zero to the *outputBufferLen* parameter specifies that the size of the output buffer is not checked. It is the responsibility of the caller to assure a sufficient output buffer size.

If NWU_UNMAPPABLE_CHAR is returned, the buffer is not filled past the unmappable character, but is valid up to that character.

By default, **NWUXLenByteToUnicodePath** converts an unmappable byte into the substitute Unicode character.

By default, **NWUXLenByteToUnicodePath** converts a special byte sequence into a Unicode character. For example, "[2620]" is converted to the single Unicode character, 0x2620.

Call **NWUXLenByteToUnicodePath** to determine the size of the string before it is converted by setting the *unicodeOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

To determine the behavior of **NWUXLenByteToUnicodePath** when an embedded NULL is encountered, see Length-Specified Byte String Conversion.

See NWUXLenByteToUnicodePath: Example.

NCP Calls

None

See Also

NWUXLenByteToUnicode, **NWUXUnicodeToUntermBytePath**

NWUXLoadByteUnicodeConverter

Locates and loads a converter to convert between Unicode and the specified code page

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLoadByteUnicodeConverter (
   nuint             codepage,
   pCONVERT N_FAR   *byteUniHandle);
```

Parameters

codepage

(IN) Specifies the code page to use. Pass zero to use the system code page.

byteUniHandle

(OUT) Points to the Unicode/byte converter handle. If an error occurs, the handle is set to NULL.

Return Values

0x00 00	SUCCESSFUL
0xFD E1	NWU_CONVERTER_NOT_FOUND
0xFD E2	NWU_TOO_MANY_FILES (DOS only)
0xFD E3	NWU_NO_PERMISSION (DOS only)
0xFD E4	NWU_OPEN_FAILED (DOS only)
0xFD E5	NWU_READ_FAILED (DOS only)
0xFD E6	NWU_OUT_OF_MEMORY

0xFD E7	NWU_CANT_LOAD_CONVERTER
0xFD E8	NWU_CONVERTER_CORRUPT

Remarks

NWUXLoadByteUnicodeConverter can be called multiple times with different code pages. This functionality allows you to perform operations involving multiple code pages. A different handle will be returned for each separate call.

Each call to **NWUXLoadByteUnicodeConverter** should have a corresponding call to **NWUXUnloadConverter** to release converter resources when they are no longer needed. To unload the appropriate converter, pass in the handle returned from the corresponding call to **NWUXLoadByteUnicodeConverter**.

If *codepage* is set to zero, the system code page will be used.

Conversions are performed according to standard Novell default behavior unless another behavior is specified through the other functions in this API set.

See NWUXLoadByteUnicodeConverter: Example.

NCP Calls

None

See Also

NWUXLoadCaseConverter, **NWUXLoadCollationConverter**, **NWUXLoadNormalizeConverter**, **NWUXUnloadConverter**

NWUXLoadCaseConverter

Locates and loads a converter to convert Unicode to upper, lower, or title case (upper case for initial letter only)

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLoadCaseConverter (
   nuint             caseFlag,
   pCONVERT N_FAR   *caseHandle);
```

Parameters

caseFlag

(IN) Specifies the flag to indicate which case converter should be loaded:

NWU_LOWER_CASE
NWU_UPPER_CASE
NWU_TITLE_CASE

caseHandle

(OUT) Points to the converter handle. If an error occurs, the handle is set to NULL.

Return Values

0x00 00	SUCCESSFUL
0xFD E1	NWU_CONVERTER_NOT_FOUND
0xFD E2	NWU_TOO_MANY_FILES (DOS only)
0xFD E3	NWU_NO_PERMISSION (DOS only)
0xFD E4	NWU_OPEN_FAILED (DOS only)

0xFD E5	NWU_READ_FAILED (DOS only)
0xFD E6	NWU_OUT_OF_MEMORY
0xFD E7	NWU_CANT_LOAD_CONVERTER
0xFD E8	NWU_CONVERTER_CORRUPT
0xFD ED	NWU_RANGE_ERROR

See NWUXLoadCaseConverter: Example.

NCP Calls

None

See Also

NWUXLoadByteUnicodeConverter, **NWUXLoadCollationConverter**, **NWUXLoadNormalizeConverter**, **NWUXUnloadConverter**

NWUXLoadCollationConverter

Locates and loads a converter to convert Unicode into collation weights

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLoadCollationConverter (
   nuint              countryCode,
   pCONVERT N_FAR   *collationHandle,);
```

Parameters

countryCode

(IN) Specifies the country code to use. Zero specifies the system country code.

collationHandle

(OUT) Points to the collation converter handle. If an error occurs, the handle is set to NULL.

Return Values

0x00 00	SUCCESSFUL
0xFD E1	NWU_CONVERTER_NOT_FOUND
0xFD E2	NWU_TOO_MANY_FILES (DOS only)
0xFD E3	NWU_NO_PERMISSION (DOS only)
0xFD E4	NWU_OPEN_FAILED (DOS only)
0xFD E5	NWU_READ_FAILED (DOS only)
0xFD E6	NWU_OUT_OF_MEMORY

0xFD E7	NWU_CANT_LOAD_CONVERTER
0xFD E8	NWU_CONVERTER_CORRUPT

See NWUXLoadCollationConverter: Example.

NCP Calls

None

See Also

NWUSGetCodePage, **NWUXLoadByteUnicodeConverter**, **NWUXLoadCaseConverter**, **NWUXLoadNormalizeConverter**, **NWUXUnloadConverter**

NWUXLoadNormalizeConverter

Locates and loads a converter to normalize Unicode characters
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLoadNormalizeConverter (
   nuint             preDeFlag,
   pCONVERT N_FAR   *normalizeHandle);
```

Parameters

preDeFlag

(IN) Specifies whether precomposed Unicode is desired:

NWU_PRECOMPOSE
NWU_DECOMPOSE

normalizeHandle

Points to the normalization converter handle (or NULL if an error occurs).

Return Values

0x00 00	SUCCESSFUL
0xFD E1	NWU_CONVERTER_NOT_FOUND
0xFD E2	NWU_TOO_MANY_FILES (DOS only)
0xFD E3	NWU_NO_PERMISSION (DOS only)
0xFD E4	NWU_OPEN_FAILED (DOS only)
0xFD E5	NWU_READ_FAILED (DOS only)
0xFD	NWU_OUT_OF_MEMORY

E6	
0xFD E7	NWU_CANT_LOAD_CONVERTER
0xFD E8	NWU_CONVERTER_CORRUPT
0xFD ED	NWU_RANGE_ERROR

Remarks

The Novell byte to Unicode converter converts by default into precomposed Unicode characters.

Decomposing results in characters being separated into base characters and following non-spacing marks wherever possible. For example, if the input contains the character U+00CC (LATIN CAPITAL LETTER I GRAVE), the output would contain U+0049 (LATIN CAPITAL LETTER I) followed by U+0300 (NON-SPACING GRAVE).

Precomposing the sequence U+0049 U+0300 would output U+00CC.

See NWUXLoadNormalizeConverter: Example.

NCP Calls

None

See Also

NWUXLoadByteUnicodeConverter, NWUXLoadCaseConverter, NWUXLoadCollationConverter, NWUXUnloadConverter

NWUXResetConverter

Resets the converter to a default state
Local Servers: to be supplied
Remote Servers: to be supplied
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXResetConverter (
   pCONVERT   convert);
```

Parameters

convert
(IN) Points to the handle of the converter to be reset.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

See NWUXResetConverter: Example.

NCP Calls

None

NWUXSetByteFunctions

Specifies the functions to be used to handle unmappable bytes and special byte sequences during byte-to-unicode conversion

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetByteFunctions (
   pCONVERT   byteUniHandle,
   NMBYTE     noMapByteFunc,
   SCBYTE     scanByteFunc,
   PRBYTE     parseByteFunc);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

noMapByteFunc

(IN) Specifies the function to be called when an unmappable byte is found.

scanByteFunc

(IN) Specifies the function to scan for special byte sequences.

parseByteFunc

(IN) Specifies the function to parse special byte sequences.

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH

Remarks

NWUXSetByteFunctions will not set the action code to NWU_CALL_HANDLER.

Values for the *noMapByteFunc*, *scanByteFunc*, and *parseByteFunc* parameters follow:

NWU_RESET_TO_DEFAULT	Resets the current setting to the default setting
NWU_UNCHANGED_FUNCTION	Leaves the current setting unchanged

See NWUXSetByteFunctions: Example.

NCP Calls

None

See Also

NWUXGetByteFunctions, **NWUXGetUniFunctions**, **NWUXGetSubByte**, **NWUXGetSubUni**, **NWUXResetConverter**, **NWUXSetNoMapAction**, **NWUXSetUniFunctions**, **NWUXSetSubByte**, **NWUXSetSubUni**

NWUXSetNoMapAction

Sets the actions to follow when an unmappable byte or an unmappable Unicode character is found

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetNoMapAction (
   pCONVERT   byteUniHandle,
   nint       noMapByteAction,
   nint       noMapUniAction);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

noMapByteAction

(IN) Specifies the action to follow for unmappable bytes during byte-to-Unicode conversion.

noMapUniAction

(IN) Specifies the action to follow for unmappable Unicode characters during Unicode-to-byte conversion.

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDED	NWU_RANGE_ERROR

Remarks

Values for either the *noMapByteAction* or *noMapUniAction* parameter are:

NWU_RETURN_ ERROR	Returns an error code of NWU_UNMAPPABLE_CHAR.
NWU_SUBSTITU TE	Uses the current substitute byte if converting to bytes, or the current substitution character if converting to Unicode characters.
NWU_CALL_ HANDLER	Calls the current NoMap handler. User defined handlers may be specified by calling the **NWUXSetByteFunctions** or **NWUXSetUniFunctions** functions.
NWU_UNCHAN GED_ ACTION	Leaves the current setting unchanged.

See NWUXSetNoMapAction: Example.

NCP Calls

None

See Also

NWUXGetNoMapAction, **NWUXGetByteFunctions**, **NWUXGetUniFunctions**, **NWUXGetSubByte**, **NWUXGetSubUni**, **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, **NWUXSetSubByte**, **NWUXSetSubUni**

NWUXSetScanAction

Enables of disables the current scan/parse functions

Local Servers: nonblocking

Remote Servers: nonblocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetScanAction (
   pCONVERT   byteUniHandle,
   nint       scanByteAction,
   nint       scanUniAction);
```

Parameters

byteUniHandle
: (IN) Points to the converter handle.

scanByteAction
: (IN) Specifies the status for current byte scan/parse functions.

scanUniAction
: (IN) Specifies the status for current Unicode scan/parse functions.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD ED	NWU_RANGE_ERROR

Remarks

Values for the *scanByteAction* or *scanUniAction* parameters follow:

NWU_ENABLED	Enables scan/parse functions
NWU_DISABLED	Disables scan/parse functions
NWU_UNCHANGED_ACTION	Leaves the current settings unchanged

See NWUXSetScanAction: Example.

NCP Calls

None

See Also

NWUXGetScanAction, **NWUXGetByteFunctions**, **NWUXGetUniFunctions**

NWUXSetSubByte

Specifies the substitution byte for the converter
Local Servers: to be supplied
Remote Servers: to be supplied
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetSubByte (
   pCONVERT   byteUniHandle,
   nuint8     substituteByte);
```

Parameters

byteUniHandle
(IN) Points to the converter handle.

substituteByte
(IN) Specifies the new substitution byte.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

Remarks

During a Unicode to byte conversion, the substitution byte is written to the output buffer if an unmappable Unicode character is found and the current action is set to NWU_SUBSTITUTE.

NWUXSetSubByte does not set the NoMap Unicode action to

NWU_SUBSTITUTE. NoMap Unicode action must be set by calling the **NWUXSetNoMapAction** function.

See NWUXSetSubByte: Example.

NCP Calls

None

See Also

NWUXGetNoMapAction, **NWUXGetByteFunctions**, **NWUXGetUniFunctions**, **NWUXGetSubByte**, **NWUXGetSubUni**, **NWUXSetNoMapAction**, **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, **NWUXSetSubUni**

NWUXSetSubUni

Specifies the substitution character for the converter

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetSubUni (
   pCONVERT   byteUniHandle,
   Unicode    subsituteUni);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

substituteUni

(OUT) Receives the new substitution character.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

Remarks

During a byte to Unicode conversion, the substitution character is written to the output buffer if an unmappable byte is found and the current NoMap byte action is set to NWU_SUBSTITUTE.

NWUXSetSubUni does not set the NoMap byte action to be followed to

NWU_SUBSTITUTE. NoMap byte action must be set by calling the **NWUXSetNoMapAction** function.

See NWUXSetSubUni: Example.

NCP Calls

None

See Also

NWUXGetNoMapAction, **NWUXGetByteFunctions**, **NWUXGetUniFunctions**, **NWUXGetSubByte**, **NWUXGetSubUni**, **NWUXSetNoMapAction**, **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, **NWUXSetSubByte**

NWUXSetUniFunctions

Specifies the functions to be used to handle unmappable Unicode characters and special Unicode sequences during unicode-to-byte conversion

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetUniFunctions (
   pCONVERT   byteUniHandle,
   NMUNI      noMapUniFunc,
   SCUNI      scanUniFunc,
   PRUNI      parseUniFunc);
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

noMapUniFunc

(IN) Specifies the Unicode function to be called when unmappable Unicode characters are found.

scanUniFunc

(IN) Specifies the function to scan for special Unicode sequences.

parseUniFunc

(IN) Specifies the function to parse special Unicode sequences.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH

Remarks

NWUXSetUniFunctions will not set the action code to NWU_CALL_HANDLER.

Values for the *noMapUniFunc*, *scanUniFunc*, and *parseUniFunc* parameters follow:

NWU_RESET_TO_DEFAULT	Resets the current setting to the default setting
NWU_UNCHANGED_FUNCTION	Leaves the current setting unchanged

See NWUXSetUniFunctions: Example.

NCP Calls

None

See Also

NWUXGetByteFunctions, **NWUXGetUniFunctions**, **NWUXGetSubByte**, **NWUXGetSubUni**, **NWUXResetConverter**, **NWUXSetNoMapAction**, **NWUXSetByteFunctions**, **NWUXSetSubByte**, **NWUXSetSubUni**

NWUXUnicodeToByte

Converts a Unicode string into a NULL-terminated byte string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToByte (
   pCONVERT    byteUniHandle,
   pnuint8     byteOutput,
   nuint       outputBufferLen,
   punicode    unicodeInput,
   pnuint      actualLength);
```

Parameters

byteUniHandle

(IN) Points to the Unicode converter handle.

byteOutput

(OUT) Points to the output buffer to receive the converted byte string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes (0 specifies that the buffer size is not checked).

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in bytes) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD	NWU_NULL_HANDLE

E9	
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EC	NWU_UNMAPPABLE_CHAR
0xFD EE	NWU_BUFFER_FULL

Remarks

By default, **NWUXUnicodeToByte** converts an unmappable Unicode character into a special 6-byte sequence. For example, 0x2620 is converted to "[2620]".

By default, **NWUXUnicodeToByte** does not recognize any special Unicode sequences for conversion. For example, the Unicode string "ab[81]cd" will be returned as the byte string "ab[81]cd".

Call **NWUXUnicodeToByte** to determine the size of the string before it is converted by setting the *byteOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXUnicodeToByte: Example.

NCP Calls

None

See Also

NWUXUnicodeToBytePath, **NWUXUnicodeToCase**, **NWUXUnicodeToCollation**, **NWUXUnicodeToNormalized**

NWUXUnicodeToBytePath

Converts a Unicode file path string into a NULL-terminated byte string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToBytePath (
   pCONVERT   byteUniHandle,
   pnuint8    byteOutput,
   nuint      outputBufferLen,
   punicode   unicodeInput,
   pnuint     actualLength);
```

Parameters

byteUniHandle

(IN) Points to the Unicode converter handle.

byteOutput

(OUT) Points to the output buffer to receive the converted byte string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes (0 specifies that the buffer size is not checked).

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in bytes) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD	NWU_NULL_HANDLE

E9	
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EC	NWU_UNMAPPABLE_CHAR
0xFD EE	NWU_BUFFER_FULL

Remarks

When you want to convert a file path, call the **NWUXUnicodeToBytePath** function rather than the **NWUXUnicodeToByte** function.

Regardless of the language being converted, **NWUXUnicodeToBytePath** interprets Unicode yen (00A5), won (20A9), backslash (005C), and the Novell defined path separator (F8F7) all as path separators and converts each to the byte backslash character (5C).

By default, **NWUXUnicodeToBytePath** converts an unmappable Unicode character into a special 6-byte sequence. For example, 0x2620 is converted to "[2620]".

By default, **NWUXUnicodeToBytePath** does not recognize any special Unicode sequences for conversion. For example, the Unicode string "ab[81]cd" will be returned as the byte string "ab[81]cd".

Call **NWUXUnicodeToBytePath** to determine the size of the string before it is converted by setting the *byteOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXUnicodeToBytePath: Example.

NCP Calls

None

See Also

NWUXSetNoMapAction, **NWUXUnicodeToByte**, **NWUXUnicodeToCase**, **NWUXUnicodeToCollation**, **NWUXUnicodeToNormalized**

NWUXUnicodeToCase

Converts a NULL-terminated Unicode string to upper case, lower case, or title case, depending on the converter pointed to

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToCase (
   pCONVERT   caseHandle,
   punicode   monocasedOutput,
   nuint      outputBufferLen,
   punicode   unicodeInput,
   pnuint     actualLength);
```

Parameters

caseHandle

(IN) Points to the monocase converter handle.

monocaseOutput

(OUT) Points to the output buffer to receive the converted Unicode string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes (0 specifies that the buffer size is not checked).

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be monocased.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL

0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EE	NWU_BUFFER_FULL

Remarks

The *caseHandle* parameter points to the converter to be used. The type of converter is specified when the converter is loaded with the **NWUXLoadCaseConverter** function.

Lower case conversions are preferred for most operations since there are more lower case than upper case characters in Unicode.

There are Unicode characters that may result in two or more Unicode characters when converted. Do not use the same buffer for both input and output strings.

Input characters that have no case are unaffected.

Call **NWUXUnicodeToCase** to determine the size of the string before it is converted by setting the *monocasedOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXUnicodeToCase: Example.

NCP Calls

None

See Also

NWUXLoadCaseConverter

NWUXUnicodeToCollation

Converts a NULL-terminated Unicode string to a string of collation weights to be used for sorting

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToCollation (
   pCONVERT    collationHandle,
   punicode    collationWeights,
   nuint       outputBufferLen,
   punicode    unicodeInput,
   pnuint      actualLength);
```

Parameters

collationHandle

(IN) Points to the collation converter handle.

collationWeights

(OUT) Points to the output buffer to receive the collation weights (optional)

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes (0 specifies that the buffer size is not checked).

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL

0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EE	NWU_BUFFER_FULL

Remarks

Weighting for collations is dependent upon the country code for which information is being collated. The country code is specific to the converter pointed to by the *collationHandle* parameter, and is specified in the *countryCode* parameter of **NWUXLoadCollationConverter** when the collation converter is loaded.

Call **NWUXUnicodeToCollation** to determine the size of the string before it is converted by setting the *collationWeights* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXUnicodeToCollation: Example.

NCP Calls

None

See Also

NWUXSetNoMapAction, **NWUXUnicodeToByte**, **NWUXUnicodeToBytePath**, **NWUXUnicodeToCase**, **NWUXUnicodeToCollation**, **NWUXUnicodeToNormalized**

NWUXUnicodeToNormalized

Converts a Unicode string to a maximal decomposed string or a minimal precomposed string, depending on the type of converter pointed to

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToNormalized (
   pCONVERT   normalizeHandle,
   punicode   normalizedOutput,
   nuint      outputBufferLen,
   punicode   unicodeInput,
   pnuint     actualLength);
```

Parameters

normalizeHandle

(IN) Points to the normalize converter handle.

normalizedOutput

(OUT) Points to the output buffer to receive the normalized output (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes (0 specifies the buffer size is not checked).

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in Unicode characters) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL

0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EE	NWU_BUFFER_FULL

Remarks

Decomposing results in characters being separated into base characters and nonspacing marks whenever possible. For example, if the input contains the character U+00CC (capital letter I grave), the output would contain U+0049 (capital letter I), followed by U+0300 (nonspacing grave).

Likewise, precomposing of the sequence U+0049 U+0300 would result in an output of U+00CC.

Call **NWUXUnicodeToNormalized** to determine the size of the string before it is converted by setting the *normalizedOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXUnicodeToNormalized: Example.

NCP Calls

None

See Also

NWUXLoadNormalizeConverter, **NWUXSetNoMapAction**, **NWUXUnicodeToByte**, **NWUXUnicodeToBytePath**, **NWUXUnicodeToCase**, **NWUXUnicodeToCollation**

NWUXUnicodeToUntermByte

Converts a Unicode string into an unterminated byte string

Local Servers: to be supplied

Remote Servers: to be supplied

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToUntermByte (
   pCONVERT    byteUniHandle,
   pnuint8     byteOutput,
   nuint       outputBufferLen,
   punicode    unicodeInput,
   pnuint      actualLength);
```

Parameters

byteUniHandle

(IN) Points to the Unicode converter handle.

byteOutput

(OUT) Points to the output buffer to receive the unterminated byte string (optional).

outputBufferLen

(IN) Specifies the maximum size of the output buffer in bytes.

unicodeInput

(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength

(OUT) Points to the returned length (in bytes) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE

0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EC	NWU_UNMAPPABLE_CHAR
0xFD EE	NWU_BUFFER_FULL

Remarks

By default, **NWUXUnicodeToUntermByte** converts an unmappable Unicode character into a special unterminated 6-byte sequence. For example, 0x2620 is converted to "[2620]".

By default, **NWUXUnicodeToUntermByte** does not recognize any special Unicode sequences for conversion. For example, the Unicode string "ab[81]cd" will be returned as the byte string "ab[81]cd".

Call **NWUXUnicodeToUntermByte** to determine the size of the string before it is converted by setting the *byteOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXUnicodeToUntermByte: Example.

NCP Calls

None

See Also

NWUXLenByteToUnicode, **NWUXUnicodeToUntermBytePath**

NWUXUnicodeToUntermBytePath

Converts a Unicode file path string into an unterminated byte string
Local Servers: to be supplied
Remote Servers: to be supplied
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToUntermBytePath (
   pCONVERT   byteUniHandle,
   pnuint8    byteOutput,
   nuint      outputBufferLen,
   punicode   unicodeInput,
   pnuint     actualLength);
```

Parameters

byteUniHandle
(IN) Points to the Unicode converter handle.

byteOutput
(OUT) Points to the output buffer to receive the unterminated byte string (optional).

outputBufferLen
(IN) Specifies the maximum size of the output buffer in bytes.

unicodeInput
(IN) Points to the input buffer containing the Unicode string to be converted.

actualLength
(OUT) Points to the returned length (in bytes) of the converted string (optional). Does not include the NULL terminator.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE

0xFD EA	NWU_BAD_HANDLE
0xFD EB	NWU_HANDLE_MISMATCH
0xFD EC	NWU_UNMAPPABLE_CHAR
0xFD EE	NWU_BUFFER_FULL

Remarks

When you want to convert a file path, call the **NWUXUnicodeToUntermBytePath** function rather than the **NWUXUnicodeToUntermByte** function.

Regardless of the language being converted, **NWUXUnicodeToUntermBytePath** interprets Unicode yen (00A5), won (20A9), backslash (005C), and the Novell defined path separator (F8F7) all as path separators and converts each to the byte backslash character (5C).

By default, **NWUXUnicodeToUntermBytePath** converts an unmappable Unicode character into a special unterminated 6-byte sequence. For example, 0x2620 is converted to "[2620]".

By default, **NWUXUnicodeToUntermBytePath** does not recognize any special Unicode sequences for conversion. For example, the Unicode string "ab[81]cd" will be returned as the byte string "ab[81]cd".

Call **NWUXUnicodeToUntermBytePath** to determine the size of the string before it is converted by setting the *byteOutput* parameter to NULL. The *outputBufferLen* parameter will be ignored, a converted string will not be returned, and the length of the string if it were converted will be returned in the *actualLength* parameter.

See NWUXUnicodeToUntermBytePath: Example.

NCP Calls

None

See Also

NWUXLenByteToUnicodePath, **NWUXUnicodeToUntermByte**

NWUXUnloadConverter

Unloads a converter and releases all associated resources
Local Servers: to be supplied
Remote Servers: to be supplied
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnloadConverter (
   pCONVERT N_FAR  *converterHandle);
```

Parameters

converterHandle
(IN) Points to the converter handle to release.

Return Values

0x00 00	SUCCESSFUL
0xFD E9	NWU_NULL_HANDLE
0xFD EA	NWU_BAD_HANDLE

Remarks

NWUXUnloadConverter should be called separately for each converter loaded with the extended Unicode functions. Exiting without unloading a converter may result in the converter remaining unnecessarily loaded. In contrast to **NWUSStandardUnicodeRelease**, **NWUXUnloadConverter** unloads a specific converter specified by the handle passed in through the *converterHandle* parameter.

See NWUXUnloadConverter: Example.

NCP Calls

None

See Also

NWUXLoadByteUnicodeConverter, **NWUXLoadCaseConverter**, **NWUXLoadCollationConverter**, **NWUXLoadNormalizeConverter**

unicat

Appends a copy of a specified string to the end of another string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unicat (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unicat
  (s1 : punicode;
   s2 : punicode
) : punicode;
```

Parameters

s1
(OUT) Points to the original string.

s2
(IN) Points to the string to be appended.

Return Values

Pointer to concatenated *s1*.

Remarks

unicat corresponds to the C **strcat** function.

The length of the resulting string is:

```
unilen (s1) + unilen (s2)
```

NCP Calls

None

See Also

unincat

unichr

Finds the first occurrence of a given character in a specified string
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unichr (
   punicode   s,
   unicode    c);
```

Pascal Syntax

```
#include <unicode.inc>

Function unichr
  (s : punicode;
   c : unicode
) : punicode;
```

Parameters

s

(IN) Points to the string to be scanned.

c

(IN) Specifies the character to be found.

Return Values

Non-N ULL	Pointer to the first occurrence of the character*c* in *s*.
NULL	*c* does not occur in*s*.

Remarks

The NULL terminator is part of the string. For example, unichr(string,0) returns a pointer to the terminating NULL character of string.

unichr corresponds to the C **strchr** function.

NCP Calls

None

See Also

unirchr, **unistr**

unicmp

Compares two Unicode strings for differences
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) unicmp (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unicmp
  (s1 : punicode;
   s2 : punicode
) : nint;
```

Parameters

s1

(IN) Points to the first string to be compared.

s2

(IN) Points to the second string to be compared.

Return Values

One of the following int values:

< 0 if *s1* is less than *s2*

= 0 if *s1* is equal to *s2*

> 0 if *s1* is greater than *s2*

Remarks

unicmp compares *s1* to *s2*, starting with the first character in each string and continuing with subsequent characters until (1) the corresponding characters differ, or (2) the end of the strings is reached. The comparison is done lexicographically, using the value of the unicode character, not the collation weight.

unicmp is useful for comparing strings for equality. Do not call **unicmp** for sorting strings into collation order.

unicmp corresponds to the C **strcmp** function.

NCP Calls

None

See Also

uniicmp, **unincmp**

unicpy

Copies from one specified string to another
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unicpy (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unicpy
  (s1 : punicode;
   s2 : punicode
) : punicode;
```

Parameters

s1
(OUT) Points to the destination string.

s2
(IN) Points to the source string.

Return Values

Pointer to *s1*.

Remarks

unicpy stops copying after it moves the terminating NULL character.

unicpy corresponds to the C **strcpy** function.

NCP Calls

None

See Also

See Also

unincpy, **unipcpy**

unicspn

Scans a specified string for the initial segment not containing any subset of the given set of characters

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(size_t) unicspn (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unicspn
  (s1 : punicode;
   s2 : punicode
) : size_t;
```

Parameters

s1

(IN) Points to the string to be scanned.

s2

(IN) Points to the character set.

Return Values

Returns the length of the initial segment of *s1* consisting entirely of characters not from *s2*.

Remarks

unicspn corresponds to the C **strcspn** function.

NCP Calls

None

See Also

unispn, **unipbrk**

uniicmp

Compares the lower case versions of two different strings (using the character values, not the collation value) for differences

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) uniicmp (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function uniicmp
  (s1 : punicode;
   s2 : punicode
) : nint;
```

Parameters

s1

(IN) Points to the first string to be compared.

s2

(IN) Points to the second string to be compared.

Return Values

One of the following int values:

< 0 if *s1* is less than *s2*

= 0 if *s1* is equal to *s2*

> 0 if *s1* is greater than *s2*

Remarks

uniicmp converts its arguments to lower case according to the standard Monocase Converter rules. It then compares *s1* to *s2*, starting with the first character in each string and continuing with subsequent characters until (1) the corresponding characters differ, or (2) the end of the strings is reached. The comparison is done lexicographically, using the value of the unicode character, not the collation weight.

uniicmp is useful for comparing strings for equality. Do not call **uniicmp** for sorting strings into collation order.

uniicmp corresponds to the C **stricmp** function.

NCP Calls

None

unilen

Returns the number of unicode characters in a string (not counting the NULL terminator)

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(size_t) unilen (
   punicode   s);
```

Pascal Syntax

```
#include <unicode.inc>

Function unilen
  (s : punicode
) : size_t;
```

Parameters

s

(IN) Points to the string for which to determine the length.

Return Values

Number of characters (not bytes) in *s*, not counting the null-terminating character.

Remarks

unilen corresponds to the C **strlen** function.

NCP Calls

None

See Also

unisize

unincat

Copies the specified number of characters from one string to the end of another string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unincat (
   punicode   s1,
   punicode   s2,
   size_t      n);
```

Pascal Syntax

```
#include <unicode.inc>

Function unincat
  (s1 : punicode;
   s2 : punicode;
   n  : size_t
) : punicode;
```

Parameters

s1

(OUT) Points to the original string.

s2

(IN) Points to the string to be appended.

n

(IN) Specifies the maximum number of characters to be appended.

Return Values

Pointer to *s1.*

Remarks

unincat copies at most *n* characters of *s2* to the end of *s1*, then appends a null character. The maximum length of the resulting string is:

```
unilen(s1)+ n
```

unincat corresponds to the C **strncat** function.

NCP Calls

None

See Also

unicat

unincmp

Compares a specified number of characters of two Unicode strings

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) unincmp (
   punicode  *s1,
   punicode  *s2,
   size_t     len);
```

Pascal Syntax

```
#include <unicode.inc>

Function unincmp
  (s1  : punicode;
   s2  : punicode;
   len : size_t
) : nint;
```

Parameters

s1

(IN) Points to the first string to be compared.

s2

(IN) Points to the second string to be compared.

n

(IN) Specifies the maximum number of characters (not bytes) to be compared.

Return Values

One of the following values:

< 0 if *s1* is less than *s2*

= 0 if *s1* is equal to *s2*

> 0 if *s1* is greater than *s2*

Remarks

unincmp compares *s1* to *s2*, for a maximum length of *n* characters, starting with the first character in each string and continuing with subsequent characters until (1) the corresponding characters differ or (2) the end of the strings is reached. The comparison is done lexicographically, using the value of the unicode character not the collation weight.

unincmp is useful for comparing strings for equality. Do not call **unincmp** for sorting strings into collation order.

unincmp corresponds to the C **strncmp** function.

NCP Calls

None

See Also

unicmp, **uninicmp**

unincpy

Copies a specified number of characters from one string to another
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unincpy (
   punicode  *s1,
   punicode   s2,
   size_t     n);
```

Pascal Syntax

```
#include <unicode.inc>

Function unincpy
  (s1 : punicode;
   s2 : punicode;
   n  : size_t
) : punicode;
```

Parameters

s1

(OUT) Points to the destination string.

s2

(IN) Points to the source string.

n

(IN) Specifies the maximum length in characters (not bytes).

Return Values

Returns a pointer to *s1*.

Remarks

unincpy copies up to the number of characters specified by the *n* parameter from the source string in the *s2* parameter to the destination string in the *s1* parameter and truncates the *s1* parameter if necessary. The destination string in the *s1* parameter might not be null-terminated if the length of the *s2* parameter is the number of characters specified by the *n* parameter or more.

Win32 clients call wcsncpy which pads any remaining space in the buffer with nulls. Non-Win32 clients do not NULLl-pad the output buffer.

unincpy corresponds to the C **strncpy** function.

NCP Calls

None

See Also

unicpy, **unipcpy**

uninicmp

Compares a specified number of characters (using the character values, not the collation value) of the lower case versions of two strings

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) uninicmp (
   punicode  *s1,
   punicode  *s2,
   size_t     len);
```

Pascal Syntax

```
#include <unicode.inc>

Function uninicmp
  (s1  : punicode;
   s2  : punicode;
   len : size_t
) : nint;
```

Parameters

s1

(IN) Points to the first string to be compared.

s2

(IN) Points to the second string to be compared.

n

(IN) Specifies the maximum number of characters (not bytes) to be compared.

Return Values

One of the following values:

< 0 if *s1* is less than *s2*

= 0 if *s1* is equal to *s2*

> 0 if *s1* is greater than *s2*

Remarks

uninicmp converts its arguments to lower case according to the standard Monocase Converter rules. It then compares *s1* to *s2*, for a maximum length of *n* characters, starting with the first character in each string and continuing with subsequent characters until (1) the corresponding characters differ or (2) the end of the strings is reached. The comparison is done lexicographically, using the value of the unicode character not the collation weight.

uninicmp is useful for comparing strings for equality. Do not call **uninicmp** for sorting strings into collation order.

uninicmp corresponds to the C **strnicmp** function.

NCP Calls

None

See Also

uniicmp

uninset

Copies a given character into the first specified number of characters of a string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) uninset (
   punicode   s,
   unicode    c,
   size_t     n);
```

Pascal Syntax

```
#include <unicode.inc>

Function uninset
  (s : punicode;
   c : unicode;
   n : size_t
) : punicode;
```

Parameters

s

(OUT) Points to the string to modify.

c

(IN) Specifies the fill character.

n

(IN) Specifies the maximum number of characters (not bytes).

Return Values

Returns a pointer to the *s* parameter.

Remarks

uninset copies the character specified by the *c* parameter into the first number of characters specified by the *n* parameter of the string pointed to by the *s* parameter. If the number specified by the *n* parameter is greater than the return value of unilen(*s*), the return value of strlen(*s*) replaces the *n* parameter.

uninset stops after (1) setting the number of characters specified by the*n* parameter, or (2) finding a NULL character.

uninset corresponds to the C**memset** function.

NCP Calls

None

See Also

uniset

unipbrk

Scans a specified string for a given set of characters
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unipbrk (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unipbrk
  (s1 : punicode;
   s2 : punicode
) : punicode;
```

Parameters

s1

(IN) Points to the string to scan.

s2

(IN) Points to the character set.

Return Values

Returns a pointer to the first occurrence of any character in the *s2* parameter. NULL is returned if no characters specified in the *s2* parameter occurs in string pointed to by the *s1* parameter.

Remarks

unipbrk scans the string pointed to by the *s1* parameter for the first occurrence of any character appearing in the *s2* parameter.

unipbrk corresponds to the C **strpbrk** function.

NCP Calls

None

See Also

unicspn, **unispn**

unipcpy

Copies one string to another up to the null-termination
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unipcpy (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unipcpy
  (s1 : punicode;
   s2 : punicode
) : punicode;
```

Parameters

s1

(OUT) Points to the destination string.

s2

(IN) Points to the source string.

Return Values

Returns a pointer to the *s1* parameter plus the return value of unilen(*s2*).

Remarks

unipcpy copies the string pointed to by the *s2* parameter to the destination string pointed to by the *s1* parameter and stops after moving the terminating NULL character.

unipcpy has no corresponding C function.

unipcpy is identical to the **unicpy** function except for the return value.

unipcpy returns a pointer to the NULL terminating character of the resulting destination string. This pointer can be used in subsequent calling **unipcpy** to concatenate a series of strings without having to scan the entire string each time.

NCP Calls

None

See Also

unicpy, **unincpy**

unirchr

Scans a string in the reverse direction, searching for the last occurrence of a given character

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unirchr (
   punicode   s,
   unicode    c);
```

Pascal Syntax

```
#include <unicode.inc>

Function unirchr
  (s : punicode;
   c : unicode
) : punicode;
```

Parameters

s

(IN) Points to the string to scan.

c

(IN) Specifies the character to find.

Return Values

Returns a pointer to the last occurrence of the character specified by the *c* parameter. NULL is returned if the character specified by the *c* parameter does not occur in the string pointed to by the *s* parameter.

Remarks

unirchr searches for the last occurence of the character specified by the *c* parameter in the string pointed to by the *s* parameter. It considers the NULL terminator to be part of the string.

unirchr corresponds to the C **strrchr** function.

NCP Calls

None

See Also

unichr

unirev

Reverses the order of all characters in a string, except for the null terminating character

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unirev (
   punicode   s);
```

Pascal Syntax

```
#include <unicode.inc>

Function unirev
  (s : punicode
) : punicode;
```

Parameters

s

(OUT) Points to the string to reverse.

Return Values

Returns a pointer to the reversed string.

Remarks

After calling **unirev**, the original contents of the string pointed to by the *s* parameter are replaced by the reversed string. The terminating NULL character is not reversed. For example, **unirev** changes "string" to "gnirts."

unirev corresponds to the C **strrev** function.

NCP Calls

None

uniset

Sets all characters in a string to a specified character
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) uniset (
   punicode    s,
   unicode     c);
```

Pascal Syntax

```
#include <unicode.inc>

Function uniset
  (s  : punicode;
   c  : unicode
)  : punicode;
```

Parameters

s
(OUT) Points to the string to modify.

c
(IN) Specifies the fill character.

Return Values

Returns a pointer to the *s* parameter.

Remarks

uniset quits when it finds the terminating NULL character.

uniset corresponds to the C **strset** function.

NCP Calls

None

See Also

See Also

uninset

unisize

Determines the size in bytes (not characters) the current string occupies including the 2-byte NULL terminator

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(size_t) unisize (
   punicode   s);
```

Pascal Syntax

```
#include <unicode.inc>

Function unisize
   (s : punicode
) : size_t;
```

Parameters

s

(IN) Points to the string for which to get the size.

Return Values

Returns the number of bytes in the string pointed to by the *s* parameter including the terminating NULL character.

NCP Calls

None

See Also

unilen

unispn

Finds the first segment of a string consisting entirely of characters from another string

NetWare Server: 4.x

Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(size_t) unispn (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unispn
  (s1 : punicode;
   s2 : punicode
) : size_t;
```

Parameters

s1

(IN) Points to the string to test.

s2

(IN) Points to the character set.

Return Values

Returns the length of the first segment of the string pointed to by the *s1* parameter that consists entirely of characters from the *s2* parameter.

Remarks

unispn corresponds to the C **strspn** function.

NCP Calls

None

See Also

unicspn, **unipbrk**

unistr

Scans a string for the first occurrence of a specified substring
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unistr (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unistr
  (s1 : punicode;
   s2 : punicode
) : punicode;
```

Parameters

s1

(IN) Points to the string to be scanned.

s2

(IN) Points to the string to be located.

Return Values

Non-N ULL	Pointer to the element in *s1*where *s2*begins (points to *s2* in *s1*.
NULL	*s2* does not occur in *s1*.

Remarks

If the *s2* parameter points to a zero length string, **unistr** returns the string pointed to by the *s1* parameter.

unistr corresponds to the C **strstr** function.

NCP Calls

None

See Also

unichr

unitok

Searches a string for tokens separated by specified delimiters
NetWare Server: 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unitok (
   punicode   s1,
   punicode   s2);
```

Pascal Syntax

```
#include <unicode.inc>

Function unitok
  (s1 : punicode;
   s2 : punicode
) : punicode;
```

Parameters

s1

(IN) Points to the string to parse.

s2

(IN) Points to the delimiter values.

Return Values

Non-N ULL	Pointer to the token found in *s1*.
NULL	No more tokens remain to be processed.

Remarks

unitok considers the string pointed to by the *s1* parameter to consist of a sequence of zero or more text tokens, separated from the delimiter pointed to by the *s2* parameter by spans of one or more characters.

The first call to **unitok** returns a pointer to the first character of the first token in the *s1* parameter and writes a NULL character into the *s1* parameter immediately following the returned token. Passing NULL for the *s1* parameter causes **unitok** to search through the string pointed to by the *s1* parameter in this way, until no tokens remain.

unitok corresponds to the C **strtok** function.

NCP Calls

None

Unicode Converter API: Guides

General Unicode Information

This section gives a very high level overview of Unicode* and explains a few of the advantages to writing to the Novell® Unicode* API.

Conversion

This section provides information about the implementation of the Novell Unicode converters and explains a few of their advantages.

Conversion Operations

Setting Up for Conversions

Byte/Unicode Conversions

Case, Collation, and Normalization Conversion

Unicode Converter API: Tasks

Collating Unicode Strings with an Extended Converter

1. **If you have not already done so, call NWUXLoadCollationConverter to initialize the converter and obtain a converter handle. For the *countryCode* parameter, pass the code for the country of the collation system to be used. Zero specifies the system country code.**

2. **Supply an output buffer for the Unicode collation weights.**

3. **Call NWUXUnicodeToCollation with the following parameter stipulations:**

   For the *collationHandle* parameter, pass the handle returned from the appropriate call to **NWUXLoadCollationConverter**.

   For the *collationWeights* parameter, pass a pointer to the output collation weights buffer.

   For the *outputBufferLen* parameter, pass length of the output buffer.

   For the *unicodeInput* parameter, pass a pointer to the Unicode input buffer.

   For the *outLength* parameter, pass the address of the pointer to the output collation weights length.

4. **Perform the necessary collation based on the output collation weights.**

5. **When the converter is not longer needed, call NWUXUnloadConverter, passing the handle returned when the converter was loaded.**

Related Topics:

Extended Unicode Collation Converter Options

Converting a String from Bytes to Unicode with a Standard Converter

NOTE: If the minimum size of the output buffer for the Unicode string is important to your application but is not known, see Performing a Standard Conversion on a String of Unknown Size.

1. **If you have not already done so, call NWUSStandardUnicodeInit.**
2. **Supply an output buffer of sufficient size to hold the output Unicode string.**
3. **Call NWUSByteToUnicode unless the string is a path. For path strings, call NWUSByteToUnicodePath. For either function do the following:**

   For the *unicodeOutput* parameter, pass a pointer to the output buffer.

   For *outputBufferLen*, pass the length of the output buffer in bytes.

   For *byteInput*, pass a pointer to the input buffer.

4. **Use the Unicode string pointed to by *unicodeOutput* as needed when NWUSByteToUnicode returns.**
5. **After all standard Unicode conversion operations are completed, call NWUSUnicodeRelease.**

Converting a String from Unicode to Bytes with a Standard Converter

Converting Path Strings with a Standard Converter

Converting a String from Unicode to Bytes with a Standard Converter

NOTE: If the required size for the output byte string buffer is not known, see Performing a Standard Conversion on a String of Unknown Size.

1. **If you have not already done so, call NWUSStandardUnicodeInit.**
2. **Supply an output buffer of sufficient size to hold the output byte string.**
3. **Call NWUSUnicodeToByte unless the string is a path. For path strings, call NWUSUnicodeToBytePath. Do the following for either function:**

   For *byteOutput*, pass a pointer to the output buffer.

   For *outputBufferLen*, pass the length of the output buffer in bytes.

   For *unicodeInput*, pass a pointer to the input buffer.

4. **When the function returns, use the Unicode string pointed to by *unicodeOutput* as needed.**

5. **After all standard Unicode conversion operations are completed, call NWUSUnicodeRelease.**

Related Topics:

Converting a String from Bytes to Unicode with a Standard Converter

Converting Path Strings with a Standard Converter

Converting Path Strings with a Standard Converter

1. **If you have not already done so, call NWUSStandardUnicodeInit.**
2. **For a Unicode-to-byte conversion, follow the instructions in Converting a String from Unicode to Bytes with a Standard Converter , but call NWUSUnicodeToBytePath instead of NWUSUnicodeToByte.**
3. **For a byte-to-Unicode conversion, follow the instructions in Converting a String from Bytes to Unicode with a Standard Converter , but call NWUSByteToUnicodePath instead of NWUSByteToUnicode.**
4. **After all standard Unicode conversion operations are completed, call NWUSUnicodeRelease.**

Related Topics:

Conversion Operations

Converting Bytes to Unicode with an Extended Converter

1. **If you have not already done so, call NWUXLoadByteUnicodeConverter. Specify the relevant code page for the *codepage* parameter.**
2. **Supply a buffer of sufficient length to hold the Unicode output string. If length is important to your application and is not known follow the steps in**
3. **Call NWUXByteToUnicode with the following parameter specifications:**

   For the *byteUniHandle* parameter, pass the handle returned from the appropriate call to **NWUXLoadByteUnicodeConverter**. Note that it is possible to have called the function more than once to make conversions for separate code pages.

For the *unicodeOutput* parameter, pass a pointer to the Unicode output string buffer.

For the *outputBufferLen* parameter, pass the maximum length of the output buffer, including the NULL terminator.

For the *byteInput* parameter, pass a pointer to the input byte buffer.

For the *actualLength* parameter, pass the address of the pointer to the output length. Note that the length pointed to on return does not include the NULL terminator.

4. **If you have called NWUXByteToUnicode to determine the required length of the output buffer, call NWUXByteToUnicode a second time. This time use *outLength* from the returned previous call to determine the required length for the output buffer.**

5. **Use the returned Unicode string as needed.**

6. **When you no longer need the converter, free it by calling NWUXUnicodeRelease and passing the relevant converter handle.**

Related Topics:

Converting Unicode to Bytes with an Extended Converter

Converting Path Strings with an Extended Converter

1. **To convert a byte path string to Unicode, follow the steps in Converting Bytes to Unicode with an Extended Converter. However, call NWUXUnicodeToBytePath instead of NWUXUnicodeToByte.**

2. **To convert a Unicode path string to bytes, follow the steps in Converting Unicode to Bytes with an Extended Converter. However, call NWUXByteToUnicodePath instead of NWUXByteToUnicode.**

Related Topics:

Converting Bytes to Unicode with an Extended Converter

Converting Unicode to Bytes with an Extended Converter

Converting Unicode String Case with a Standard Converter

1. **If you have not already done so, call NWUSStandardUnicodeInit.**

2. **Supply a buffer of sufficient length to receive the lower or upper case**

Unicode string.

3. **Call NWUSUnicodeToLowerCase or NWUSUnicodeToUpperCase for the actual conversion.**

   For the *lowerCaseOutput* or *upperCaseOutput* parameter, pass a pointer to the output buffer.

   For the *outputBufferLen* parameter, pass a pointer to the length of the output buffer.

   For the *unicodeInput* parameter, pass a pointer to the Unicode string input buffer.

4. **After all standard Unicode conversion operations are completed, call NWUSUnicodeRelease.**

Related Topics:

Conversion Operations

Converting Unicode String Case with an Extended Converter

1. **If you have not already done so, call NWUXLoadCaseConverter to initialize the converter and obtain a converter handle.**

   NOTE: You must specify the case to which the converter is to convert in the *caseFlag* parameter of **NWUXLoadCaseConverter**. If an initialized converter is not set to the desired case, initialize another case converter. It is possible to have multiple converters, each set to a different option (upper case, lower case, title case).

2. **Supply a buffer of sufficient size to hold the output monocased Unicode string. In some languages, changing the case of a string may change the string length.**

3. **Call NWUXUnicodeToCase with the following parameter stipulations:**

   For the *caseHandle* parameter, pass the handle returned from appropriate call to **NWUXLoadCaseConverter**.

   For the *monocaseOutput* parameter, pass the address of the output buffer for the Unicode monocase string.

   For the *outputBufferLen* parameter, pass the length of the output buffer.

   For the *unicodeInput* parameter, pass a pointer to the input string buffer.

4. **Use the monocased output string as needed.**

5. **When the converter is no longer needed, call NWUXUnloadConverter, passing the handle returned when the converter was loaded.**

Related Topics:

Extended Unicode Case Converter Options

Variable Unicode Casing

Converting Unicode to Bytes with an Extended Converter

1. **If you have not already done so, call NWUXLoadByteUnicodeConverter. Specify the relevant code page for the *codepage* parameter.**

2. **Supply a buffer of sufficient length to hold the byte output string. If you don't know that length, see**

3. **Call NWUXUnicodeToByte with the following parameter specifications:**

   For the *byteUniHandle* parameter, pass the handle returned from the appropriate call to **NWUXLoadByteUnicodeConverter**. Note that it is possible to have called the function more than once to make conversions for separate code pages.

   For the *byteOutput* parameter, pass a pointer to the byte output string buffer.

   For the *outputBufferLen* parameter, pass the maximum length of the output buffer, including the NULL terminator.

   For the *unicodeInput* parameter, pass a pointer to the input Unicode buffer.

   For the *outLength* parameter, pass the address of the pointer to the output length. Note that the length pointed to on return does not include the NULL terminator.

4. **If you have called NWUXUnicodeToByte to determine the required length of the output buffer, call NWUXUnicodeToByte a second time. This time use *outLength* from the returned previous call to determine the required length for the output buffer.**

5. **Use the returned byte string as needed.**

6. **When you no longer need the converter, free it by calling NWUXUnicodeRelease and passing the relevant converter handle.**

Related Topics:

Converting Bytes to Unicode with an Extended Converter

Determining Output String Length with an Extended Converter

NOTE: Although the example in this task is written to convert from byte to Unicode, the same general procedure can be used for determining output string length for any extended conversion.

1. **If you have not done so already, call the appropriate function to load the required converter:**

   NWUXLoadByteUnicodeConverter

   NWUXLoadCaseConverter

   NWUXLoadCollationConverter

   NWUXLoadNormalizeConverter

2. **With the converter loaded, call the appropriate conversion function once to determine the required length of the output buffer. Pass a NULL for the output buffer parameter.**

   ```
   NWUXByteToUnicode(converter, NULL, 0, inbuf,
   &actualLength);
   ```

   When the function returns, the*actualLength* parameter points to the output string length.

   NOTE: The length pointed to is only the number of characters in the string. **It does not include a NULL terminator.**

3. **Add one the returned *actualLength* for the NULL terminator. If the output is to be a Unicode string, multiply *actualLength* by sizeof(unicode) to get the required number of bytes. Then allocate memory.**

   ```
   bufsiz=acutalLen+1;

   outbuf=(punicode)malloc(bufsize*sizeof(unicode));
   ```

4. **Do the real conversion.**

   ```
   NWUXByteToUnicode(converter, outbuf, bufsize, inbuf,
   NULL);
   ```

5. **When the converter is no longer needed, free the output buffer and call NWUXUnloadConverter as explained in Unloading Converters.**

Related Topics:

Converting Bytes to Unicode with an Extended Converter

Converting Unicode to Bytes with an Extended Converter

Finding Out Which Code Page a Standard Converter Uses

The standard Unicode API set does not enable you to specify a code page or a country code. Instead the functions in the set make use of the code page and country code specified by the operating system. Through this task you can find out which code page and country code the operating system specifies.

1. **If you have not already done so, call NWUSStandardUnicodeInit.**
2. **Call NWUSGetCodePage. On return, the *pCodePage* parameter points to the local code page and the *pCountry* parameter points to the local country code.**
3. **After all standard Unicode conversion operations are completed, call NWUSUnicodeRelease.**

Related Topics:

Initializing a Standard Unicode Converter

Converting a String from Bytes to Unicode with a Standard Converter

Converting a String from Unicode to Bytes with a Standard Converter

Handling Unmappable Characters with an Extended Converter

By default, the extended Unicode API uses the Default Conversion Behavior. However, the developer can change that process in several ways:

Change the NoMap action

Change the Substitution character

Change the handler function

Change scan/parse functions

1. **If you want to make a change, then reset functions to prechange behavior:**

   Before making the change, call the **NWUXGet..** version of the relevant function and save the return.

Call the **NWUXSet..** version to make the change.

Restore previous settings by calling the **NWUXSet..** version a second time and passing the values returned from the **NWUXGet..** version.

2. **To change the NoMap action, call NWUXSetNoMapAction, and set the *noMapByteAction* or *noMapUniAction* to**

   `NWU_RETURN_ERROR`

   `NWU_SUBSTITUTE` or

   `NWU_CALL_HANDLER`

   Set either of these parameters to `NWU_UNCHANGED_ACTION` if no change is needed.

3. **The change the substitution character, call NWUXSetSubByte or NWUXSetSubUni and set the *substituteByte* or *substituteUni* parameter to the new substitution character.**

4. **To change the function handler, call NWUXSetByteFunctions or NWUXSetUniFunctions and set the *noMapByteFunc* or *noMapUniFunc* parameter to point to the new function.**

5. **To return all settings to the system defaults, call NWUXResetConverter and pass the handle of the converter to return to defaults.**

Related Topics:

Converting Bytes to Unicode with an Extended Converter

Converting Unicode to Bytes with an Extended Converter

Setting Substitution Characters with an Extended Converter

Setting Scan/Parse Functions with an Extended Converter

Unmappable Characters

Substitution Characters

Scan/Parse Action

NoMap, Scan, and Parse Functions

Initializing a Standard Unicode Converter

1. **Call NWUXStandardUnicodeInit.**

Because the default settings are no subject to change for the standard API

Because the default settings are no subject to change for the standard API set, this function has no arguments. On completion, the converter is set up to convert byte/Unicode strings in either direction using the local system code page, as well as converting the case of Unicode strings.

Related Topics:

Standard Unicode Functions

Initializing an Extended Unicode Converter

1. **Choose the converter needed for your current task. Choices include the following:**

   Byte/Unicode Converter (Converts byte strings to Unicode and Unicode strings to bytes)

   Unicode Case Converter (Converts Unicode strings to upper, lower, or title case)

   Unicode Collation Converter (Assigns collation weights to Unicode strings)

   Unicode Normalization Converter (Converts Unicode strings between precomposed and decomposed forms)

2. **Choose the options needed for the current operation.**

   Extended Byte/Unicode Converter Options

   Extended Unicode Case Converter Options

   Extended Unicode Collation Converter Options

   Extended Unicode Normalization Converter Options

3. **Call the appropriate load function.**

4. **Use the converter for the needed operations.**

5. **When you no longer need the converter, free it by calling NWUXUnloadConverter.**

Related Topics:

Converting Bytes to Unicode with an Extended Converter

Converting Unicode to Bytes with an Extended Converter

Converting Unicode String Case with an Extended Converter

Collating Unicode Strings with an Extended Converter

Normalizing Unicode Strings with an Extended Converter

Normalizing Unicode Strings with an Extended Converter

1. **If you have not already done so, call NWUXLoadNormalizeConverter to initialize the converter and obtain a converter handle. For the *preDeFlag* parameter, pass either NWU_PRECOMPOSE or NWU_DECOMPOSE. You can initialize one converter for each mode if you need to.**

2. **Supply a buffer for the output normalized Unicode string. If the output buffer size is important to your application and is not known, follow the instructions in Determining Output String Length with an Extended Converter.**

3. **Call NWUXUnicodeToNormalized with the following parameter stipulations:**

   For the *normalizeHandle* parameter, pass the handle returned from appropriate call to **NWUXLoadNormalizeConverter**.

   For the *NormalizedOutput* parameter, pass the address of the output buffer for the Unicode normalized string.

   For the *outputBufferLen* parameter, pass the length of the output buffer.

   For the *unicodeInput* parameter, pass a pointer to the input string buffer.

4. **Use the normalized output string as needed.**

5. **When the converter is not longer needed, call NWUXUnloadConverter, passing the appropriate normalization converter handle.**

Related Topics:

Extended Unicode Normalization Converter Options

Normalization Unicode API Functions

Performing a Standard Conversion on a String of Unknown Size

1. **If you have not already done so, call NWUSStandardUnicodeInit.**

2. **Call the appropriate conversion function, but set the output buffer to NULL. The *outputBufferLen* parameter is then ignored.**

```
NWUSByteToUnicode(NULL, 0, byteInput,
&actualLength);
```

3. **Add one to the returned *actualLength* for the NULL terminator. If the output is to be a Unicode string, multiply *actualLength* by sizeof(unicode) to get the required number of bytes. Then allocate memory.**

   ```
   bufsiz=actualLength+1;
   ```

   ```
   outbuf=(punicode)malloc(bufsize*sizeof(unicode));
   ```

4. **Do the real conversion.**

   ```
   NWUSByteToUnicode(outbuf, bufsize, byteInput, NULL);
   ```

5. **After all standard Unicode conversion operations are completed, call NWUSUnicodeRelease.**

Related Topics:

Conversion Operations

Setting Scan/Parse Functions with an Extended Converter

1. **If you have not already done so, call NWUXLoadByteUnicodeConverter. Specify the relevant code page for the *codepage* parameter.**

2. **If you want to return to the current settings after a change, call the Get version of the relevant function and save the return before calling the Set function to make the change.**

3. **The enable or disable scan/parse functions, call NWUXSetScanAction and set either the *scanByteAction* or the *scanUniAction* parameter to**

   `NWU_ENABLED` or

   `NWU_DISABLED`

   Pass NWU_UNCHANGED to either parameter that requires no change.

   The default is disabled for Unicode-to-byte conversion and enabled for byte-to-Unicode conversion.

4. **To set either a scan or parse (or both) function other than the default system functions, call NWUXSetByteFunctions or NWUXSetUniFunctions and pass a pointer to the new function(s). Pass `NWU_UNCHANGED_FUNCTION` to pointers in these functions for which not change is needed.**

Related Topics:

Scan/Parse Action

NoMap, Scan, and Parse Functions

Setting Substitution Characters with an Extended Converter

Substitution characters can be used as one option to replace unmappable characters in either Unicode-to-byte or byte-to-Unicode conversions. The other options are to return an error or to call a handler function.

1. **If you have not already done so, call NWUXLoadByteUnicodeConverter to initialize the converter and obtain a converter handle.**
2. **If you need to save and restore the original substitution character, call the appropriate function to obtain current substitution character information:**

   For Unicode-to-byte conversions, call **NWUXGetSubByte**.

   For byte-to-Unicode conversions, call **NWUXGetSubUni**.

   Save the returned character for later restoration.
3. **Call the appropriate function to set a new substitution character:**

   For Unicode-to-byte conversions, call **NWUXSetSubByte**.

   For byte-to-Unicode conversions, call **NWUXSetSubUni**.

   For either function, pass the converter handle and the new substitution character.
4. **Convert the strings as needed.**
5. **When the most recently set substitution characters are no longer needed, reset as appropriate:**

   To reset the previous substitution character values, call **NWUXSetSubByte** or **NWUXSetSubUni**, passing the values returned from Step 2 above.

   To reset default values, call **NWUXResetConverter**, and pass the converter handle.
6. **When the converter is not longer needed, call NWUXUnloadConverter, passing the appropriate converter handle.**

Related Topics:

Substitution Characters

Substitution Characters

Conversion Control

Unloading Converters

Standard converters can be initiated multiple times by one or more applications. Each time a converter is initiated, it should be released when conversion operations are complete.

1. **To release a standard converter, call NWUSStandardUnicodeRelease. The function has no parameters.**

Extended converters perform a variety of conversions, but each is nloaded in the same way.

1. **To unload an extended converter, call NWUXUnloadConverter and pass the handle of the converter to be unloaded.**

Related Topics:

Initializing/Loading Unicode Converters

Unicode Converter API: Concepts

Conversion Control

The standard converter allows only for Default Conversion Behavior, and the byte/Unicode converters uses that behavior as a starting default. However, the extended API allows the developer the following choices:

action for Unmappable Characters

specification of Substitution Characters

specification of Scan/Parse Action

Developers can set options other than the system defaults through **NWUXSetByteFunctions**, **NWUXSetUniFunctions**, and **NWUXSetNoMapAction**.

Related Topics:

Default Conversion Behavior

Handling Unmappable Characters with an Extended Converter

Setting Scan/Parse Functions with an Extended Converter

Setting Substitution Characters with an Extended Converter

Default Conversion Behavior

For Unicode-to-byte and byte-to-Unicode conversion,the following behavior is automatic for standard functions and is default for extended functions. Standard functions provide for this behavior only, but extended functions allow extensive modification.

Unicode-to-byte Conversion

Unmappable Unicode characters result in a call to a function handler, which forms the basis of lossless round trip conversion. The handler converts each unmappable Unicode character into a string of six byte characters as follows:

Unmappable Unicode character U+NNNN becomes byte string "[NNNN]".

For example, if the character "#" is an unmappable Unicode "skull and crossbones" character (U+2620),

the Unicode input string "abc#def"

converts to the local byte output string "abc[2620]def".

Scan/parse functions are disabled.

Byte-to-Unicode Conversion

Unmappable byte characters result in a substitution by the standard Unicode REPLACEMENT CHARACTER---0xFFFD.

The **scan/parse** functions are enabled, reversing the Unicode-to-byte function handler behavior. These scan/parse functions scan for the byte sequence "[NNNN]", where NNNN is a string of four hexadecimal digits. Scan/parse convert each such sequence to a single Unicode character whose value is U+NNNN. For example, if the character "#" were again the Unicode "skull and crossbones" character (U+2620),

the local byte input string "abc[2620]def"

converts to the Unicode output string "abc#def".

Related Topics:

Standard Unicode Functions

Lossless Conversion

Extended Actions for Unmappable Characters

With the extended Unicode API functions, you can select any of three actions when an unmappable character is encountered:

Return an error

Convert unmappable characters into a substitution character (default or user-defined)

Call a handler function (default or user-defined)

For example, if a Unicode string is being converted to local code page in order to be displayed, a user-defined handler function could convert an unmappable character into a red blinking question mark. The default handler inserts the hex value of the unmappable character enclosed in square brackets in place of the character, as explained in Default Conversion Behavior.

Related Topics:

Conversion Control

Extended Byte/Unicode Converter Options

Extended byte/Unicode converters can convert either from Unicode to bytes or from bytes to Unicode, depending on the function called after the converter is loaded. Variable converter options include the code page and the country code, specified in the parameters of **NWUXLoadByteUnicodeConverter**.

Related Topics:

Supported Code Pages

Conversion Control

Extended Unicode Case Converter Options

Case converter options are set with the*caseFlag* parameter of include the following possibilities:

Constant	**Result**
NWU_LOWER_CASE	Converts a Unicode string to all lower case
NWU_UPPER_CASE	Converts a Unicode string to all upper case
NWU_TITLE_CASE	Converts the first letter of each word in a Unicode string to upper case

Related Topics:

Conversion Control

Converting Unicode String Case with an Extended Converter

Extended Unicode Collation Converter Options

The only variable option for the collation converter is the country code. This option is set in with the*countryCode* parameter of **NWUXLoadCollationConverter**. Multiple converters can be loaded, each with a different code page.

Related Topics:

Conversion Control

Collating Unicode Strings with an Extended Converter

Extended Unicode Functions

The following functions offer more flexibility and choices that the standard Unicode functions offer.

Function	Header File	Comment
NWUXByteToUnicode	nunicode.h	Converts a NULL-terminated byte string into a Unicode string.
NWUXByteToUnicodePath	nunicode.h	Converts a NULL-terminated file path byte string into a Unicode string.
NWUXGetByteFunctions	nunicode.h	Returns the functions used for handling unmappable bytes and special byte sequences during byte-to-Unicode conversion.
NWUXGetCharSize	nunicode.h	Returns the character size (1 or 2) of the next character in the byte string.
NWUXGetNoMapAction	nunicode.h	Returns the actions to follow when an unmappable byte sequence and an unmappable Unicode character are found.
NWUXGetScanAction	nunicode.h	Gets the status of current scan/parse functions.
NWUXGetUniFunctions	nunicode.h	Returns the functions used for handling unmappable Unicode characters and special Unicode sequences during Unicode-to-byte conversion.
NWUXGetSubByte	nunicode.h	Returns the substitution byte for the converter pointed to.
NWUXGetSubUni	nunicode.h	Returns the current substitution Unicode character for the converter.
NWUXLenByteToUnicode	nunicode.h	Converts a length-specified byte string into a NULL-terminated Unicode string.

NWUXLenUnicodeToByte	nunicode.h	Converts a length-specified Unicode string into a NULL-terminated byte string.
NWUXLoadByteUnicodeConverter	nunicode.h	Locates and loads a converter to convert between Unicode and the specified code page.
NWUXLoadCaseConverter	nunicode.h	Locates and loads a converter to convert Unicode to upper, lower, or title case (upper case for initial letter only).
NWUXLoadCollationConverter	nunicode.h	Locates and loads a converter to convert Unicode into collation weights.
NWUXLoadNormalizeConverter	nunicode.h	Locates and loads a converter to normalize Unicode characters.
NWUXResetConverter	nunicode.h	Resets the converter to a default state.
NWUXSetNoMapAction	nunicode.h	Sets the actions to follow when an unmappable byte or an unmappable Unicode character is found.
NWUXSetByteFunctions	nunicode.h	Specifies the functions to be used to handle unmappable bytes and special byte sequences during byte-to-Unicode conversion.
NWUXSetScanAction	nunicode.h	Enables or disables the current scan/parse functions.
NWUXSetSubByte	nunicode.h	Specifies the substitution byte for the converter.
NWUXSetSubUni	nunicode.h	Specifies the substitution character for the converter.
NWUXSetUniFunctions	nunicode.h	Specifies the functions to be used to handle unmappable Unicode characters and special Unicode sequences during Unicode-to-byte conversion.
NWUXUnicodeToByte	nunicode.h	Converts a Unicode string into a NULL-terminated byte string.

NWUXUnicodeToBytePath	nunicode.h	Converts a Unicode file path string into a NULL-terminated byte string.
NWUXUnicodeToUntermBytePath	nunicode.h	Converts a Unicode string into a an unterminated byte string.
NWUXUnicodeToUntermBytePath	nunicode.h	Converts a Unicode file path string into an unterminated byte string.
NWUXUnicodeToCase	nunicode.h	Converts a NULL-terminated Unicode string to upper case, lower case, or title case, depending on the converter pointed to.
NWUXUnicodeToCollation	nunicode.h	Converts a NULL-terminated Unicode string to a string of collation weights to be used for sorting.
NWUXUnicodeToNormalized	nunicode.h	Converts a Unicode string to a maximal decomposed string or a minimal precomposed string, depending on the type of converter pointed to.
NWUXUnloadConverter	nunicode.h	Unloads a converter and releases all associated resources.

Related Topics:

Standard Unicode Functions

Old and New Unicode Header Files

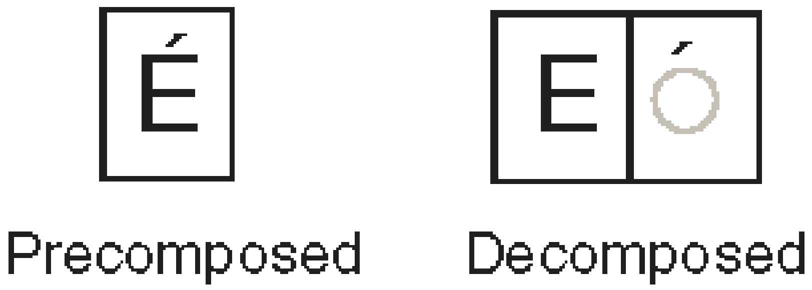
Extended Unicode Normalization Converter Options

Normalization converters can be opened with one of the following options specified:

NWU_PRECOMPOSED

NWU_DECOMPOSED

Normalization API functions allow a Unicode string to be converted to pre-composed or decomposed form. Precomposed characters consist of a character with an accompanying diacritical mark, like an "E" with an acute accent. The decomposed form consists two characters---an ordinary character, like a plain "E" and a nonspacing diacritical character, such as an acute accent. The following graphic illustrates this difference:



In the Unicode numbering scheme, the precomposed character and each element of the decomposed character have separate numeric representations. However, the elements of the decomposed form have no necessary numeric relationship to the precomposed form. For example, the characters in the above illustration have the following Unicode values:

Character	**Unicode Value**
É	U+00C9
E	U+0045
´ [nonspacing]	U+0301

Related Topics:

Normalizing Unicode Strings with an Extended Converter

Normalization Unicode API Functions

Initializing/Loading Unicode Converters

NWUSStandardUnicodeInit must be called before using any of the standard converter functions. Each call to **NWUSStandardUnicodeInit** should have a corresponding call to **NWUSStandardUnicodeRelease** when the conversion operations are complete.

NWUSStandardUnicodeInit automatically loads a byte/Unicode converter for the native systems code page, an uppercase converter, and a lower case converter.

One of the **NWUXLoad...** functions must be called to load an extended converter. The load functions return a separate handle to each converter loaded. That handle must be passed to any other extended converter functions involving the respective converter. Each **NWUXLoad...** function called should be followed with a corresponding call to **NUWXUnloadConverter** when the converter is no longer needed.

Related Topics:

Initializing a Standard Unicode Converter

Initializing an Extended Unicode Converter

Unloading Converters

Length-Specified Byte String Conversion

Unicode provides functions for converting a specified number of bytes from a byte string into Unicode characters.

For the standard converter, the functions are **NWUSLenByteToUnicode** and **NWUSLenByteToUnicodePath**.

For extended converters, the functions are **NWUXLenByteToUnicode** and **NWUXLenByteToUnicodePath**.

These functions behave exactly like their "Len" counterparts: **NWUSByteToUnicode**, **NWUSByteToUnicodePath**, **NWUXByteToUnicode**, and **NWUXByteToUnicodePath**, with the following exceptions:

Each "Len" functions allows a developer to specify the exact number of bytes to be converted.

Each of the "Len" functions can have an unterminated string for the input buffer.

If the length-specified function encounters a NULL before the specified number of bytes have been converted, it stops converting and returns NWU_EMBEDDED_NULL. However, it converts the bytes prior to the NULL, and returns the number of Unicode characters converted.

For example, consider the byte string `abcdefgh` for the following example:

```
ccode = NWUSLenByteToUnicode (&outbuf, MAX_LEN, inbuf,
5, &outlen);
```

On return `ccode` is zero, `outbuf` contains the Unicode string `abcde`, and `outlen` contains 5.

In contrast, given the byte string `abc\0defg`, the **NWUSLenByteToUnicode** function returns NWU_EMBEDDED_NULL.

On return `outbuf` contains the Unicode string `abc` and `outlen` contains 3.

Related Topics:

Converting a String from Bytes to Unicode with a Standard Converter

Converting Bytes to Unicode with an Extended Converter

Unterminated Byte Strings from Unicode Conversion

Lossless Conversion

Many Unicode characters cannot be represented in a given local code page. However, situations arise when a Unicode string is converted to a local byte string, then converted back to Unicode. With the former Unicode API, any unmappable characters were lost in this process. The new Unicode API functions provide the capability to convert from Unicode to local and back to Unicode without losing any information.

Related Topics:

Conversion Control

Microsoft Path Separator Discrepancies

Differences between Novell and Microsoft Unicode translations tables in different languages have sometimes caused Unicode path strings to be stored with different path separators. Novell Unicode path conversion API functions called from any language recognize these differences and correctly convert any Unicode path separator back to the local path separator character.

Related Topics:

Converting Path Strings with a Standard Converter

Converting Path Strings with an Extended Converter

Multiple Code Page Converters

Previously, an open code page had to be closed before a new code page could be opened. Using the new extended API functions, you can have multiple byte/Unicode converters loaded and active simultaneously, each with a different code page. For each converter, a handle is returned when the load fuction completes.

It is important to unload each converter when it is no longer needed, as explained in Unloading Converters. This helps avoid the possibility of tying up system resources needlessly.

up system resources needlessly.

Related Topics:

Initializing an Extended Unicode Converter

Unicode Converter API Implementation

Instead of building on Unicode tables contained in data files, in this API builds on converters implemented as DLLs. These converter files may be placed in any directory where the system searches for DLLs (for example, C:\WINDOWS\SYSTEM).

Converter files follow a naming convention that designates both the converter type and the supported platform. The format of that convention is UNI_[TYP].[PLT], where [TYP] is the converter type and [PLT] is the supported platform. Extensions to designate supported platforms are as follows:

.W32---Windows 95 and NT

.W16---Windows 3.1

.DLL---OS/2 (OS/2 requires ".DLL" extension)

.NLM--- NLM platform

There are 4 types of converters. All illustrations that follow use ".W32," the extension for Windows 95 and NT.

Byte/Unicode converters convert both from byte to Unicode and from Unicode to byte. These converters are designated in the naming convention by the numeric value of the code page.

For example, UNI_1252.W32 is the converter for code page 1252 (W95/NT).

Case converters convert cases from one Unicode string to another Unicode string. They are designated in the naming convention as follows:

UNI_MON.W32---Lowercasing

UNI_UPR.W32---Uppercasing

UNI_TTL.W32---Titlecasing

Collation converters collate Unicode strings according to the collation conventions of a specified country. Collation converters are designated in the naming convention by the letter "C" followed by the country code of the specified country.

For example, UNI_C1.W32 is the collation converter for country code 1 (US).

Normalization converters convert to precomposed or decomposed Unicode characters. They are designated in the naming convention as follows:

UNI_PRE.W32---Precomposing converter

UNI_DEC.W32---Decomposing converter

When an "extended" converter is opened, a handle is returned which is used in subsequent calls to extended converter functions. The developer may change various options for a particular converter without affecting other extended converters.

Once the standard converter is opened, it may be used by any number of programs. The developer cannot change preset standard converter options.

Related Topics:

Conversion Control

NoMap, Scan, and Parse Functions

NWUXSetByteFunctions and **NWUXSetUniFunctions** set the NoMap, Scan, and Parse functions for the extended converter. The NoMap function is enabled if the NoMapAction is set to NWU_CALL_HANDLER, and the Scan and Parse functions are enabled if the ScanAction option is set to NWU_ENABLED.

The default behavior (the only behavior available for the standard converter) is to use the system supplied UniNoMap function and the ByteScan/Parse functions as described in Default Conversion Behavior. These functions implement round-trip conversion from Unicode to byte to Unicode.The developer may replace any of these functions with custom versions.

Related Topics:

Scan/Parse Action

Setting Scan/Parse Functions with an Extended Converter

Normalization Unicode API Functions

The following functions are used for normalization:

NWUXLoadNormalizeConverter	Locates and loads a converter to normalize Unicode characters.

NWUXUnicodeTo Normalized	Converts a Unicode string to a maximal decomposed string or a minimal precomposed string, depending on the type of converter pointed to.

NWUXLoadNormalizeConverter can be set to point to one of the following converters:

NWU_PRECOMPOSED

NWU_DECOMPOSED

Related Topics:

Extended Unicode Normalization Converter Options

Normalizing Unicode Strings with an Extended Converter

Old and New Unicode Header Files

The header file nunicode.h replaces unicode.h to define the function prototypes, return codes, and so forth. The Unicode utility functions (such as **unicpy** and **unicat**) are defined in both headers. Therefore, use nunicode.h for files using the new Unicode API functions, and unicode.h for files using the old Unicode API functions.

NOTE: Do not mix old and new APIs in the same file, although they may be mixed in different files of the same process or thread.

Related Topics:

Standard Unicode Functions

Extended Unicode Functions

Scan/Parse Action

Two scan action options are defined, one for converting Unicode-to-byte, and one for converting byte-to-Unicode. In the extended API, options are enabled or disabled through **NWUXSetScanAction**. By default, the scanAction is disabled for Unicode-to-byte and enabled byte-to-Unicode.

Enabling the option causes an automatic prescan of the input string for any special sequences and calls a parse function to replace such sequences with something else in the output string.

When the scanAction option is enabled, a Scan function is called internally to scan the input string before the conversion. If it finds a special sequence, the conversion is performed up to that point and then the Parse function is called internally. The functions are never called directly by the developer.

Rather, they are set as explained in NoMap, Scan, and Parse Functions.

The system supplies default scan and parse functions for both byte-to-Unicode and Unicode-to-byte conversions. The byte-to-Unicode scan/parse functions operate as described in Default Conversion Behavior ---where # is the Unicode "skull and crossbones character" (U+2620), the byte input string "abc[2620]def" becomes the Unicode output string "abc#def".

By default, scan/parse action is disabled for Unicode-to-byte conversion because the need for such action is very rare. If it is enabled, it operates in a similar way to byte-to-Unicode conversion. It scans for two hexadecimal digits surrounded by square brackets in a Unicode input string and converts them into a byte character of the same hexadecimal value in the byte output string.

Related Topics:

NoMap, Scan, and Parse Functions

Setting Scan/Parse Functions with an Extended Converter

Standard Unicode Functions

These functions allow for conversions between byte and Unicode strings and between different kinds of Unicode strings. These standard functions use only the converters supplied by Novell, and do not allow for other than default behavior.

Function	Header File	Comment
NWUSByteToUnicode	nunicode.h	Converts a NULL-terminated byte string into a Unicode string.
NWUSByteToUnicodePath	nunicode.h	Converts a NULL-terminated file path bytestring into a Unicode string.
NWUSGetCodePage	nunicode.h	Returns the code page used to specify which converters are loaded.
NWUSLenByteToUnicode	nunicode.h	Converts a length-specified byte string into a NULL-terminated Unicode string.
NWUSLenUnicodeToByte	nunicode.h	Converts a length-specified Unicode string into a NULL-terminated byte

		string.
NWUSStandardUnicodeInit	nunicode.h	Loads the converters needed for the standard Unicode functions.
NWUSStandardUnicodeRelease	nunicode.h	Releases all resources allocated by the standard converter
NWUSUnicodeToByte	nunicode.h	Converts a Unicode string into a NULL-terminated byte string.
NWUSUnicodeToBytePath	nunicode.h	Converts a file path Unicode string into a NULL-terminated byte string.
NWUSUnicodeToUntermByte	nunicode.h	Converts a Unicode string into an unterminated byte string.
NWUSUnicodeToUntermBytePath	nunicode.h	Converts a file path Unicode string into an unterminated byte string.
NWUSUnicodeToLowerCase	nunicode.h	Converts a NULL-terminated Unicode string to Unicode lower case characters.
NWUSUnicodeToUpperCase	nunicode.h	Converts a NULL-terminated Unicode string to Unicode upper case characters.

Related Topics:

Standard and Extended Unicode API Functions

Extended Unicode Functions

Standard and Extended Unicode API Functions

The newer Unicode API contains two sets of functions--- "standard" and "extended" functions.

"Standard" API functions, which begin with **NWUS-**, provide a simple interface using the platform's native country and code page. They follow standard conversion behavior options, such as how to handle an unmappable character. Standard functions follow a default conversion behavior that is not subject to adjustment by the developer.

"Extended" API functions, which begin with **NWUX-**, allow the country code and code page to be specified, and allow extensive control over conversion options.

Related Topics:

Standard Unicode Functions

Extended Unicode Functions

Substitution Characters

If the NoMapAction is set to NWU_SUBSTITUTE, a substitute byte or Unicode character is output when an unmappable character is encountered. By default, NWU_SUBSTITUTE is set for Unicode-to-byte conversion and not set for byte-to-Unicode conversion.

The default substitution byte or Unicode character is determined by the converter, since different countries often have different preferences on what to display for undefined characters. For byte-to-Unicode conversion, the substitution character is U+FFFD, designated as the Unicode REPLACEMENT character. For Unicode-to-byte conversions, the converters generally set the default substitution byte to 0x03.

You can find out what the substitution characters is through **NWUXGetSubByte** or **NWUXGetSubUni**. You can set a new substituting character through **NWUXSetSubByte** or **NWUXSetSubUni**.

Related Topics:

Unmappable Characters

Setting Substitution Characters with an Extended Converter

Supported Code Pages

The following shows the code pages supported by Novell.

Table auto. Code Page Support

Country or Language	OEM CP	ANSI CP
U.S., parts of South America and parts of Europe	437	1252
Canada, South America, Western Europe, U. S.	850	1252

Slavic	852	1250
Cyrillic	855	1251
Turkish	857	1254
Portutuese	860	1252
Icelandic	861	1252
Hebrew	862	1255
Canadian French	863	1252
Arabic	864	1256
Scandanvia	865	1252
Russia	866	1251
Greek	869	1253
Thai	874	874
Japan	897	
Japan	932	932
PRC	936	936
Korea	949	949
Taiwan, Hong Kong	950	950

Related Topics:

Initializing an Extended Unicode Converter

Unmappable Characters

This option defines what action to take when an unmappable Unicode character or (less likely) an unmappable byte is encountered during conversion. The options are to

return an error,

use a substitution character,

or call a handler function.

These options are set for individually for Unicode-to-byte conversion and byte-to-Unicode conversion. **NWUXGetNoMapAction** and **NWUXSetNoMapAction** specify both options. Refer to the function reference for details.

It is important to note that the default for Unicode-to-byte conversion is to call the handler function. That handler is described in Default Conversion Behavior.

Related Topics:

Unicode Converter API Implementation

Conversion Control

Unterminated Byte Strings from Unicode Conversion

Four functions in this Unicode API conversion set provide for unterminated byte string output from Unicode input---**NWUSUnicodeToUntermByte**, **NWUSUnicodeToUntermBytePath**, **NWUXUnicodeToUntermByte**, and **NWUXUnicodeToUntermBytePath**.

These functions are identical to **NWUSUnicodeToByte**, **NWUSUnicodeToBytePath**, **NWUXUnicodeToByte**, and **NWUXUnicodeToBytePath** with one exception---the output byte string is unterminated. A trailing zero is not appended to the converted byte string.

In all other details---kinds of values that can be passed, operations performed, and numeric values returned---the above two sets of functions are identical.

Related Topics:

Converting a String from Bytes to Unicode with a Standard Converter

Converting Bytes to Unicode with an Extended Converter

Length-Specified Byte String Conversion

Variable Unicode Casing

Previous versions of the Novell Unicode API converted code page strings only into lower case Unicode strings. This limitation is now removed so that Unicode strings are no longer limited to lower case. Unicode strings now be converted to upper case or lower case with standard functions, and upper, lower, or or title case (first letter of each word is capitalized) with the extended functions.

Related Topics:

Converting Unicode String Case with a Standard Converter

Converting Unicode String Case with an Extended Converter

Extended Unicode Case Converter Options

What is Unicode?

NOTE: This is a very short introduction to Unicode*. Much more is published and available from standard computer information sources.

Unicode is a standard for character representation designed to accommodate every character in every language that is likely to be used in any computer application. Representation includes alphabetic, ideographic, and symbolic characters. Developed by companies that collectively constitute the Unicode Consortium, the standard uses a numbering system similar to ASCII characters, but has some fundamental differences. Most importantly, Unicode uses 16 bits for each character. That feature has several positive results, three of which follow:

Over 65,000 characters can be represented, enough for every character of nearly every language in use today

Each Unicode character is associated with a single, unique hexadecimal number equivalent to the 16-bit value of the character

Unicode eliminates the need for state checks and interrupts when an application changes from one language to another or mixes characters from multiple languages

Related Topics:

Why Use Unicode?

Standard and Extended Unicode API Functions

Standard Unicode Functions

Extended Unicode Functions

Why Use Unicode?

Several advantages make it wise to incorporate Unicode into your programming practices.

1. **Because all NDS strings are stored in Unicode format, applications enabled for NDS must use Unicode strings.**

   NDS is increasingly being accepted as an industry standard, providing a rapidly expanding market for NDS enabled solutions. All strings and paths in NDS are stored in Unicode format, so strings in such solutions must be stored in or convertible to Unicode. This is true for all applications, whether they are designed to be used internationally or not. Using Unicode is also a requirement of applications that take advantage of present or future Novell services based on NDS, such as Novell Distributed Print Services (NDPS™) and catalog services.

2. **Unicode simplifies or eliminates many challenges associated with**

double-byte characters.

Since all Unicode characters are uniformly 16 bits long, Unicode eliminates the need to distinguish between single-byte and double-byte characters. This has at least two advantages:

Moving from character to character is simply a matter of incrementing or decrementing.

Unicode eliminates the need for special functions and precautions that prevent landing between bytes of a double-byte character.

3. Since all Unicode characters are in the same set, **Unicode makes it possible to mix characters from widely differing languages and separate code pages.**
4. As an industry standard, **Unicode increases the ability of an application to run (and thus be marketed) in a variety of countries**.

Related Topics:

What is Unicode?