Novell Developer Kit

www.novell.com

UNICODE

October 5, 2005

Novell®

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export, or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. Please refer to www.novell.com/info/exports/ for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Copyright © 1993-2005 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.novell.com/company/legal/patents/ and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell developer products, and to get updates, see developer.novell.com/ndk. To access online documentation for Novell products, see www.novell.com/documentation.

Novell Trademarks

AppNotes is a registered trademark of Novell, Inc.
AppTester is a registered trademark of Novell, Inc., in the United States.
ASM is a trademark of Novell, Inc.
BorderManager is a registered trademark of Novell, Inc.
BrainShare is a registered service mark of Novell, Inc., in the United States and other countries.
C3PO is a trademark of Novell, Inc.
Certified Novell Engineer is a service mark of Novell, Inc.
Client32 is a trademark of Novell, Inc.
CNE is a registered service mark of Novell, Inc.
ConsoleOne is a registered trademark of Novell, Inc.
Controlled Access Printer is a trademark of Novell, Inc.
Custom 3rd-Party Object is a trademark of Novell, Inc.
DeveloperNet is a registered trademark of Novell, Inc., in the United States and other countries.
DirXML is a registered trademark of Novell, Inc.
eDirectory is a trademark of Novell, Inc.
Excelerator is a trademark of Novell, Inc.
exteNd is a trademark of Novell, Inc.
exteNd Director is a trademark of Novell, Inc.
exteNd Workbench is a trademark of Novell, Inc.
FAN-OUT FAILOVER is a trademark of Novell, Inc.
GroupWise is a registered trademark of Novell, Inc., in the United States and other countries.
Hardware Specific Module is a trademark of Novell, Inc.
Hot Fix is a trademark of Novell, Inc.
iChain is a registered trademark of Novell, Inc.
Internetwork Packet Exchange is a trademark of Novell, Inc.
IPX is a trademark of Novell, Inc.
IPX/SPX is a trademark of Novell, Inc.
jBroker is a trademark of Novell, Inc.
Link Support Layer is a trademark of Novell, Inc.
LSL is a trademark of Novell, Inc.
ManageWise is a registered trademark of Novell, Inc., in the United States and other countries.
Mirrored Server Link is a trademark of Novell, Inc.
Mono is a registered trademark of Novell, Inc.
MSL is a trademark of Novell, Inc.
My World is a registered trademark of Novell, Inc., in the United States.
NCP is a trademark of Novell, Inc.
NDPS is a registered trademark of Novell, Inc.
NDS is a registered trademark of Novell, Inc., in the United States and other countries.
NDS Manager is a trademark of Novell, Inc.
NE2000 is a trademark of Novell, Inc.
NetMail is a registered trademark of Novell, Inc.
NetWare is a registered trademark of Novell, Inc., in the United States and other countries.
NetWare/IP is a trademark of Novell, Inc.
NetWare Core Protocol is a trademark of Novell, Inc.
NetWare Loadable Module is a trademark of Novell, Inc.

NetWare Management Portal is a trademark of Novell, Inc.

NetWare Name Service is a trademark of Novell, Inc.

NetWare Peripheral Architecture is a trademark of Novell, Inc.

NetWare Requester is a trademark of Novell, Inc.

NetWare SFT and NetWare SFT III are trademarks of Novell, Inc.

NetWare SQL is a trademark of Novell, Inc.

NetWire is a registered service mark of Novell, Inc., in the United States and other countries.

NLM is a trademark of Novell, Inc.

NMAS is a trademark of Novell, Inc.

NMS is a trademark of Novell, Inc.

Novell is a registered trademark of Novell, Inc., in the United States and other countries.

Novell Application Launcher is a trademark of Novell, Inc.

Novell Authorized Service Center is a service mark of Novell, Inc.

Novell Certificate Server is a trademark of Novell, Inc.

Novell Client is a trademark of Novell, Inc.

Novell Cluster Services is a trademark of Novell, Inc.

Novell Directory Services is a registered trademark of Novell, Inc.

Novell Distributed Print Services is a trademark of Novell, Inc.

Novell iFolder is a registered trademark of Novell, Inc.

Novell Labs is a trademark of Novell, Inc.

Novell SecretStore is a registered trademark of Novell, Inc.

Novell Security Attributes is a trademark of Novell, Inc.

Novell Storage Services is a trademark of Novell, Inc.

Novell, Yes, Tested & Approved logo is a trademark of Novell, Inc.

Nsure is a registered trademark of Novell, Inc.

Nterprise is a trademark of Novell, Inc.

Nterprise Branch Office is a trademark of Novell, Inc.

ODI is a trademark of Novell, Inc.

Open Data-Link Interface is a trademark of Novell, Inc.

Packet Burst is a trademark of Novell, Inc.

PartnerNet is a registered service mark of Novell, Inc., in the United States and other countries.

Printer Agent is a trademark of Novell, Inc.

QuickFinder is a trademark of Novell, Inc.

Red Box is a trademark of Novell, Inc.

Red Carpet is a registered trademark of Novell, Inc., in the United States and other countries.

Sequenced Packet Exchange is a trademark of Novell, Inc.

SFT and SFT III are trademarks of Novell, Inc.

SPX is a trademark of Novell, Inc.

Storage Management Services is a trademark of Novell, Inc.

SUSE is a registered trademark of SUSE AG, a Novell business.

System V is a trademark of Novell, Inc.

Topology Specific Module is a trademark of Novell, Inc.

Transaction Tracking System is a trademark of Novell, Inc.

TSM is a trademark of Novell, Inc.

TTS is a trademark of Novell, Inc.

Universal Component System is a registered trademark of Novell, Inc.

Virtual Loadable Module is a trademark of Novell, Inc.

VLM is a trademark of Novell, Inc.

Yes Certified is a trademark of Novell, Inc.

ZENworks is a registered trademark of Novell, Inc., in the United States and other countries.

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

This guide describes Unicode, its functions, and features:

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation.

Documentation Updates

For the most recent version of this guide, see NLM and NetWare Libraries for C (including CLIB and XPlat) (http://developer.novell.com/ndk/clib.htm).

Additional Information

For information about other CLib and XPlat interfaces, see the following guides:

- NLM Development Concepts, Tools, and Functions (http://developer.novell.com/ndk/doc/clib/ndev_enu/data/hqgkbkjr.html)
- Program Management (http://developer.novell.com/ndk/doc/clib/prog_enu/data/h9qu926c.html)
- NLM Threads Management (http://developer.novell.com/ndk/doc/clib/thmp_enu/data/h7g6q8vc.html)
- Connection, Message, and NCP Extensions (http://developer.novell.com/ndk/doc/clib/cmgnxenu/data/hvxfva0i.html)
- Multiple and Inter-File Services (http://developer.novell.com/ndk/doc/clib/mlti_enu/data/hby40vgi.html)
- Single and Intra-File Services (http://developer.novell.com/ndk/doc/clib/sngl_enu/data/h68b1qom.html)
- Volume Management (http://developer.novell.com/ndk/doc/clib/vol__enu/data/h6ixme3w.html)
- Server Management (http://developer.novell.com/ndk/doc/clib/srvr_enu/data/hzvjztxz.html)
- Client Management (http://developer.novell.com/ndk/doc/clib/clnt_enu/data/he77rked.html)
- Network Management (http://developer.novell.com/ndk/doc/clib/nwrk_enu/data/hvyko5s2.html)
- Internationalization (http://developer.novell.com/ndk/doc/clib/intl_enu/data/h70a28iu.html)
- Sample Code (http://developer.novell.com/ndk/doc/clib/code_enu/data/hwtnc7wc.html)

- Getting Started with NetWare Cross-Platform Libraries for C (http://developer.novell.com/ndk/doc/clib/startenu/data/hv9aw5v8.html)
- Bindery Management (http://developer.novell.com/ndk/doc/clib/bind_enu/data/h9qzn7u5.html)

For CLib and XPlat source code projects, visit Forge (http://forge.novell.com).

For help with CLib and XPlat problems or questions, visit the NLM and NetWare Libraries for C (including CLIB and XPlat) Developer Support Forums (http://developer.novell.com/ndk/devforums.htm). There are two for NLM development (XPlat and CLib) and one for Windows XPlat development.

Documentation Conventions

In this documentation, a greater-than symbol (>) is used to separate actions within a step and items within a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

Concepts

1

Unicode is divided into two components:

- 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. These functions are defined in nunicode.h.

- 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 unicode.h file contains these functions, which automatically implement the necessary translation tables that are contained in the uni_*.w16 and uni_*.w32 files.

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, Unicode Converter offers important advantages. We suggest that you begin using it for new development.

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

1.1 What is Unicode?

This is a very short introduction to Unicode. Complete information 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

1.2 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.
5. 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.

6. 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.)

1.3 Unicode Converter and Unicode Table Header Files

We recommend using the new Unicode Converter functions in for new development efforts. For more information see Section 1.6, “Unicode Converter,” on page 16 and “Tasks” on page 31.

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 Unicode Converter functions, and unicode.h for files using the old Unicode functions.

NOTE: Do not mix Unicode Table and Unicode Converter in the same file, although they may be mixed in different files of the same process or thread.

1.4 Supported Code Pages

The following shows the code pages supported by Novell.

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
Portuguese	860	1252
Icelandic	861	1252
Hebrew	862	1255
Canadian French	863	1252
Arabic	864	1256
Scandinavia	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

1.5 Unicode String Functions

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

Function	Header File	Equivalent Function
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.
uniicmp	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.

Function	Header File	Equivalent Function
uninicmp	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.h	Corresponds to the standard strstr.
unitok	unicode.h	Corresponds to the standard strtok.

1.6 Unicode Converter

The Unicode converter supports the following features:

1.6.1 Standard and Extended Unicode Converter Functions

The newer Unicode contains two sets of functions- "standard" and "extended" functions.

"Standard" 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" functions, which begin with NWUX*, allow the country code and code page to be specified, and allow extensive control over conversion options.

Standard Unicode Converter 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	Description
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.

Extended Unicode Converter Functions

The following functions offer more flexibility and choices that the standard Unicode functions offer.

Function	Header File	Description
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.

Function	Header File	Description
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).
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.

Function	Header File	Description
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.
NWUXUnloadConverter	nunicode.h	Unloads a converter and releases all associated resources.

1.6.2 Unicode Converter Implementation

Unicode Converter is based on a set of 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
- .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. The [TYP] component of the converter file name is the number specifying the desired code page.

  For example, UNI_1252.W32 is the converter for code page 1252 (W95/NT).
- *Case converters* convert cases in one Unicode string to different cases in another Unicode string. The [TYP] component of the converter file name is MON for lowercasing, UPR for uppercasing, and TTL for titlecasing (first letter of each word capitalized). For example
  - 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. The [TYP] component of the converter file name is 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. The [TYP] component of the converter file name is PRE for converting to precomposed characters and DEC for converting to decomposed characters:

- 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.

In contrast, once the standard converter is opened, it may be used by any number of programs. The developer cannot change preset standard converter options.

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".

Conversion Control

The standard converter allows only for Default Conversion Behavior (page 20), and the byte/Unicode converters uses that behavior as a starting default. However, the extended functions allow you the following choices:

- action for “Unmappable Characters” on page 24

- specification of "Substitution Characters" on page 24
- specification of "Scan/Parse Action" on page 25

You can set options other than the system defaults by calling NWUXSetByteFunctions (page 152), NWUXSetUniFunctions (page 162), and NWUXSetNoMapAction (page 154).

For more information, see:

- "Handling Unmappable Characters with an Extended Converter" on page 39
- "Setting Scan/Parse Functions with an Extended Converter" on page 41
- "Setting Substitution Characters with an Extended Converter" on page 40

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 unicode Converter API functions provide the capability to convert from Unicode to local and back to Unicode without losing any information.

Supported Code Pages

Section 1.4, "Supported Code Pages," on page 14 shows the code pages supported by Novell.

1.6.3 Conversion Operations

Although a standard Unicode converter behaves in fundamentally the same way on every platform this API supports, access to converters by any specific application can vary.

NetWare-global variables are global to the entire system. A standard converter initialized with NWUSStandardUnicodeInit (page 105) or NWUSStandardUnicodeOverride (page 107) is the only standard converter available to any application on the platform. If the converter is changed through a call to NWUSStandardUnicodeOverride, the change also affects any application requesting standard conversions.

Windows 95, Windows 98, and Windows NT -global variables are global only to a single process. Each process in which a thread calls NWUSStandardUnicodeInit (page 105) or NWUSStandardUnicodeOverride (page 107) gets its own copy of the global variables associated with the standard converter. It is therefore possible for one process to perform Unicode conversions with the system default codepage converter and for another process to perform conversions with an explicitly specified converter. Each process that calls a standard converter must also release the converter and its associated resources with a corresponding call to NWUSStandardUnicodeRelease (page 109).

For related information, see "Unicode Converter Implementation" on page 19.

Location of Converter DLLs

Converters in the Unicode Converter API set are installed during the installation process.

NWUSStandardUnicodeInit automatically loads a byte/Unicode converter for the native system 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.

For related information, see:

- “Initializing a Standard Unicode Converter” on page 33
- “Loading an Extended Unicode Converter” on page 33
- “Unloading Converters” on page 34

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 converters for the following kinds of conversions:

- Byte-to-Unicode and Unicode-to-byte conversions between the Unicode set and the native system code page
- Conversions of Unicode strings to all upper case
- Conversions of Unicode strings to all lower case

Other kinds of conversions require one or more extended converters and the functions in the extended (NWUX...) set.

Each of the extended converters is called with a separate NWUXLoad... function, and each such call returns a handle that is specific to the converter loaded. That handle is then passed to any other extended functions that require the respective converter. When an extended converter is no longer needed, it should be unloaded with a call to NUWXUnloadConverter.

For related information, see:

- “Initializing a Standard Unicode Converter” on page 33
- “Loading an Extended Unicode Converter” on page 33
- “Unloading Converters” on page 34

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.

For related information, see:

- "Converting Bytes to Unicode with a Standard Converter" on page 35
- "Converting Bytes to Unicode with an Extended Converter" on page 37
- "Length-Specified Byte String Conversion" on page 25

NWU_CONVERTER_NOT_FOUND Error

If the NWU_CONVERTER_NOT_FOUND error is returned when a standard converter is being initialized or an extended converter is being loaded, the converter DLL was not found in any of the expected locations. The system looks for the converter DLL (or NLM) in the usual system DLL/NLM search path. Where the system searches depends upon the operating system under which the application operates.

For example, in Windows 32 applications, the search order is

1. The directory from which the application was loaded
2. The current directory
3. Varies between Windows 95 and Windows NT:
   - For Windows 95, the Windows system directory
   - For Windows NT

     The 32-bit Windows system directory
     The 16-bit Windows system directory
4. The Windows directory
5. Each component in the PATH variable

For NLM applications, the search order is

1. The NLM search paths
2. C: \NWSERVER
3. C: \

1.6.4 Byte/Unicode Conversions

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.

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 (page 147).

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” on page 20.

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 function completes.

It is important to unload each converter when it is no longer needed, as explained in “Unloading Converters” on page 34. This helps avoid the possibility of tying up system resources needlessly.

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 (page 132) and NWUXSetNoMapAction (page 154) 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” on page 20.

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 (page 136) or NWUXGetSubUni (page 138). You can set a new substituting character through NWUXSetSubByte (page 158) or NWUXSetSubUni (page 160).

Fore related information, see:

- “Unmappable Characters” on page 24

- “Setting Substitution Characters with an Extended Converter” on page 40

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 (page 156). 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” on page 25.

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” on page 20—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.

For related information, see“Setting Scan/Parse Functions with an Extended Converter” on page 41

NoMap, Scan, and Parse Functions

NWUXSetByteFunctions (page 152) and NWUXSetUniFunctions (page 162) 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” on page 20. These functions implement round-trip conversion from Unicode to byte to Unicode.The developer may replace any of these functions with custom versions.

For related information, see “Setting Scan/Parse Functions with an Extended Converter” on page 41

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.

For related information, see

- “Converting Bytes to Unicode with a Standard Converter” on page 35
- “Converting Bytes to Unicode with an Extended Converter” on page 37
- “Unterminated Byte Strings from Unicode Conversion” on page 22

1.6.5 Case, Collation, and Normalization Conversion

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

For related information, see: “Converting Unicode String Case with an Extended Converter” on page 42

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 title case (first letter of each word is capitalized) with the extended functions.

For related information, see

- “Converting Unicode String Case with a Standard Converter” on page 41
- “Converting Unicode String Case with an Extended Converter” on page 42

1.7 Unicode Table Concepts

We recommend using the new Unicode Converter API functions in for new development efforts. For more information see Section 1.6, “Unicode Converter,” on page 16 and “Tasks” on page 31.

1.7.1 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.

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

Section 1.4, “Supported Code Pages,” on page 14 shows the code pages supported by Novell.

1.7.2 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 the specific function reference.

1.7.3 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.

1.7.4 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.

1.7.5 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.

Tasks

2

This documentation describes common tasks associated with Unicode Converter.

2.1 Migrating from Unicode Table to Unicode Converter

1 Include the Unicode Converter header file instead of the Unicode Table header file.

Unicode Table API	Unicode Converter API
#include <unicode.h>	#include <nunicode.h>

NOTE: The Unicode Table API and the Unicode Converter API can be mixed in the same application, but not in the same file.

2 For the majority of conversions, replace Unicode Table functions with Standard Unicode Converter functions.

Unicode Table API	Unicode Converter API
NWInitUnicodeTables	NWUSStandardUnicodeInit
NWFreeUnicodeTables	NWUSStandardUnicodeRelease
NWGetCollationHandle	No longer needed. No handle argument is required for the Standard Unicode API functions.
NWGetLocalToUnicodeHandle	
NWGetMonocasehandle	
NWGetUnicodeToLocalHandle	
NWLocalToUnicode	One of the following, depending on whether the input string is a path or not, and whether the input is null-terminated or length-specified: NWUSByteToUnicode NWUSByteToUnicodePath NWUSLenByteToUnicode NWUSLenByteToUnicodePath

Unicode Table API	Unicode Converter API
NWUnicodeToLocal	One of the following, depending on whether the input string is a path or not, and whether the output string should be null-terminated or unterminated:
	NWUSUnicodeToByte
	NWUSUnicodeToBytePath
	NWUSUnicodeToUntermByte
	NWUSUnicodeToUntermBytePath
NWUnicodeToMonocase	NWUSUnicodeToLowerCase
	NWUSUnicodeToUpperCase

3 For the more complex conversions, replace Unicode Table functions with Extended Unicode Converter functions.

Unicode Table API	Unicode Converter API
NWLoadRuleTable	One of the following, depending on what type of conversion is needed:
	NWUXLoadByteUnicodeConverter
	NWUXLoadCaseConverter
NWUnloadRuleTable	NWUXUnloadConverter

4 To change the default substitution character, or the default behavior for handling unmappable characters or round-trip conversion, use Extended Converter instead of Standard Converter.

5 Use the same unicode string manipulation functions as before (for example, unicat, unicpy, etc.). Note that uniicmp and uninicmp (case insensitive comparison routines) automatically use the new lowercase converters when nunicode.h is specified.

For related information, see

- "Initializing a Standard Unicode Converter" on page 33
- "Loading an Extended Unicode Converter" on page 33
- "Converting Bytes to Unicode with a Standard Converter" on page 35
- "Converting Unicode to Bytes with a Standard Converter" on page 35
- "Converting Bytes to Unicode with an Extended Converter" on page 37
- "Converting Unicode to Bytes with an Extended Converter" on page 37
- "Handling Unmappable Characters with an Extended Converter" on page 39
- "Unloading Converters" on page 34

2.2 Setting Up for Conversions

Conversion setup consists of the following tasks:

- Section 2.2.1, “Initializing a Standard Unicode Converter,” on page 33
- Section 2.2.2, “Loading an Extended Unicode Converter,” on page 33
- Section 2.2.3, “Changing Standard Unicode Converters,” on page 34
- Section 2.2.4, “Unloading Converters,” on page 34

2.2.1 Initializing a Standard Unicode Converter

1. Call NWUSStandardUnicodeInit (page 105).

   On completion, the local system code page converter is set up to convert byte strings to Unicode or Unicode strings to bytes, or to convert the case of Unicode strings.
2. To initialize a standard converter for a code page other than the system default, call NWUSStandardUnicodeOverride (page 107) and pass the numeric value of the desired code page for the `codepage` parameter.

NOTE: Access to global variables initialized with a standard converter can vary according to platform, as explained in “Conversion Operations” on page 21.

When standard conversions are completed, call to NWUSStandardUnicodeRelease to free resources associated with NWUSStandardUnicodeInit or NWUSStandardUnicodeOverride.

For related information, see:

- “Finding Out Which Code Page a Standard Converter Uses” on page 36
- “Standard Unicode Converter Functions” on page 16

2.2.2 Loading 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” on page 23
   - “Case, Collation, and Normalization Conversion” on page 26
3. Call the appropriate load function.
   - NWUXLoadByteUnicodeConverter (page 147)
   - NWUXLoadCaseConverter (page 149)
4. Use the converter for the needed operations.
5. When the converter is no longer needed, free it by calling NWUXUnloadConverter (page 174).

For related information, see:

- "Converting Bytes to Unicode with an Extended Converter" on page 37
- "Converting Unicode to Bytes with an Extended Converter" on page 37
- "Converting Unicode String Case with an Extended Converter" on page 42

2.2.3 Changing Standard Unicode Converters

1 Call NWUSStandardUnicodeOverride (page 107), passing the numeric value of the desired code page for the `codepage`. If a standard converter is not currently loaded, NWUSStandardUnicodeOverride initializes the specified code page converter and sets up the global variables for standard Unicode conversions.

NOTE: Access to global variables initialized with a standard converter can vary according to platform, as explained in "Conversion Operations" on page 21.

For related information, see:

- "Converting Bytes to Unicode with a Standard Converter" on page 35
- "Converting Unicode to Bytes with a Standard Converter" on page 35
- "Finding Out Which Code Page a Standard Converter Uses" on page 36
- "Initializing a Standard Unicode Converter" on page 33

2.2.4 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 (page 109). The function has no parameters.

Extended converters perform a variety of conversions, but each is unloaded in the same way.

1 To unload an extended converter, call NWUXUnloadConverter (page 174) and pass the handle of the converter to be unloaded.

For related information, see "Initializing/Loading Unicode Converters" on page 22

2.3 Standard Conversion Tasks

The following are some common tasks you need to perform when using a standard converter:

- Section 2.3.1, "Converting Bytes to Unicode with a Standard Converter," on page 35
- Section 2.3.2, "Converting Unicode to Bytes with a Standard Converter," on page 35
- Section 2.3.3, "Converting Path Strings with a Standard Converter," on page 35
- Section 2.3.4, "Performing a Standard Conversion on a String of Unknown Size," on page 36
- Section 2.3.5, "Finding Out Which Code Page a Standard Converter Uses," on page 36

2.3.1 Converting 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" on page 36.

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 (page 95) or NWUSLenByteToUnicode (page 101) unless the string is a path. For path strings, call NWUSByteToUnicodePath (page 97) or NWUSLenByteToUnicodePath (page 103).
   - 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.
   - For `inLength` of the *Len* functions, pass the length of the input string in bytes (might not include a NULL terminator).
4. Use the Unicode string pointed to by `unicodeOutput` as needed when the function returns.
5. After all standard Unicode conversion operations are completed, call NWUSStandardUnicodeRelease.

- "Converting Unicode to Bytes with a Standard Converter" on page 35

2.3.2 Converting 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" on page 36.

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 (page 110) or NWUSUnicodeToUntermByte (page 116) unless the string is a path. For path strings, call NWUSUnicodeToBytePath (page 112) or NWUSUnicodeToUntermBytePath (page 118).
   - 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 NWUSStandardUnicodeRelease.

For related information, see "Converting Bytes to Unicode with a Standard Converter" on page 35

2.3.3 Converting Path Strings with a Standard Converter

1. If you have not done so already, call NWUSStandardUnicodeInit.

2 Convert as explained in “Converting Bytes to Unicode with a Standard Converter” on page 35 or “Converting Unicode to Bytes with a Standard Converter” on page 35, but call one of the following path-specific functions:

- NWUSByteToUnicodePath (page 97)
- NWUSLenByteToUnicodePath (page 103)
- NWUSUnicodeToBytePath (page 112)
- NWUSUnicodeToUntermBytePath (page 118)

2.3.4 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 NWUSStandardUnicodeRelease.

2.3.5 Finding Out Which Code Page a Standard Converter Uses

1 Call NWUSGetCodePage (page 99). On return, the `pCodePage` parameter points to the code page and the `pCountry` parameter points to the country code used by the standard converter currently loaded. If a standard converter is not currently loaded, NWUSGetCodePage returns the code page and country code that will be used when NWUXStandardUnicodeInit is called.

For related information, see:

- “Initializing a Standard Unicode Converter” on page 33
- “Converting Bytes to Unicode with a Standard Converter” on page 35
- “Converting Unicode to Bytes with a Standard Converter” on page 35

2.4 Extended Conversion Tasks

The following are some common tasks you need to perform when using an extended converter:

- Section 2.4.1, “Converting Bytes to Unicode with an Extended Converter,” on page 37
- Section 2.4.2, “Converting Unicode to Bytes with an Extended Converter,” on page 37
- Section 2.4.3, “Converting Path Strings with an Extended Converter,” on page 38

- Section 2.4.4, “Determining Output String Length with an Extended Converter,” on page 38
- Section 2.4.5, “Handling Unmappable Characters with an Extended Converter,” on page 39
- Section 2.4.6, “Setting Substitution Characters with an Extended Converter,” on page 40
- Section 2.4.7, “Setting Scan/Parse Functions with an Extended Converter,” on page 41

2.4.1 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 (page 122) or NWUXLenByteToUnicode (page 142) unless the string is a path. For path strings, call NWUXByteToUnicodePath (page 124) or NWUXLenByteToUnicodePath (page 144)
   - 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 `inLength` of the *Len* functions, pass the length of the input string in bytes (might not include a NULL terminator).
   - 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.

For related information, see “Converting Unicode to Bytes with an Extended Converter” on page 37

2.4.2 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 “Determining Output String Length with an Extended Converter” on page 38.
3. Call NWUXUnicodeToByte (page 164) 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.

For related information, see "Converting Bytes to Unicode with an Extended Converter" on page 37

2.4.3 Converting Path Strings with an Extended Converter

1. If you have not done so already, call NWUXLoadByteUnicodeConverter.
2. Convert as explained in "Converting Bytes to Unicode with an Extended Converter" on page 37 or "Converting Unicode to Bytes with an Extended Converter" on page 37, but call one of the following path-specific functions:
   - NWUXByteToUnicodePath (page 124)
   - NWUXLenByteToUnicodePath (page 144)
   - NWUXUnicodeToBytePath (page 166)
   - NWUXUnicodeToUntermBytePath (page 172)

2.4.4 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
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" on page 34.

For related information, see:

- "Converting Bytes to Unicode with an Extended Converter" on page 37
- "Converting Unicode to Bytes with an Extended Converter" on page 37

2.4.5 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 To make a change, then reset functions to pre-change behavior:

- Before making the change, call the NWUXGet.. version of the relevant function (identified in steps 2 through 5 below), and save the return.
- Call the NWUXSet.. version to make the change.
- After performing conversions with the changed setting, restore pre-change settings by calling the NWUXSet.. version again and passing the values returned from the call to NWUXGet...

2 To change the NoMap action, call NWUXSetNoMapAction (page 154), 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 (page 158) or NWUXSetSubUni (page 160) and set the `substituteByte` or `substituteUni` parameter to the new substitution character.

4. To change the function handler, call NWUXSetByteFunctions (page 152) or NWUXSetUniFunctions (page 162) and set the `noMapByteFunc` or `noMapUniFunc` parameter to point to the new function.
5. To return all settings to the system defaults, call NWUXResetConverter (page 151) and pass the handle of the converter.

For related information, see:

- "Converting Bytes to Unicode with an Extended Converter" on page 37
- "Converting Unicode to Bytes with an Extended Converter" on page 37
- "Setting Substitution Characters with an Extended Converter" on page 40
- "Setting Scan/Parse Functions with an Extended Converter" on page 41
- "Unmappable Characters" on page 24
- "Substitution Characters" on page 24
- "Scan/Parse Action" on page 25
- "NoMap, Scan, and Parse Functions" on page 25

2.4.6 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 (page 136).
   - For byte-to-Unicode conversions, call NWUXGetSubUni (page 138).
   - 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 (page 158).
   - For byte-to-Unicode conversions, call NWUXSetSubUni (page 160).
   - 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 (page 158) or NWUXSetSubUni (page 160), passing the values returned from Step 2 above.
   - To reset default values, call NWUXResetConverter (page 151), and pass the converter handle.
6. When the converter is not longer needed, call NWUXUnloadConverter, passing the appropriate converter handle.

For related information, see:

- "Substitution Characters" on page 24
- "Conversion Control" on page 20

2.4.7 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 (page 156) 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 (page 152) or NWUXSetUniFunctions (page 162) and pass a pointer to the new function(s). Pass NWU_UNCHANGED_FUNCTION to pointers in these functions for which not change is needed.

For related information, see:

- "Scan/Parse Action" on page 25
- "NoMap, Scan, and Parse Functions" on page 25

2.5 Case, Collation, and Normalization Conversion

The following sections explain how to convert the case of the Unicode string:

- Section 2.5.1, "Converting Unicode String Case with a Standard Converter," on page 41
- Section 2.5.2, "Converting Unicode String Case with an Extended Converter," on page 42

2.5.1 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 (page 114) or NWUSUnicodeToUpperCase (page 120) 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 NWUSStandardUnicodeRelease.

2.5.2 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 (page 168) 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.

For related information, see "Standard and Extended Unicode Converter Functions" on page 16.

Examples

3

This documentation provides an example of how to use Unicode Table functions.

3.1 Unicode Operations

The following code uses Internationalization services to find a country ID and code page. The example illustrates how NDS 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
 *                the unicode tables before using NDS.
 *
 * 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_page);
     printf("\nReturn code from NWInitUnicodeTables is %X\n", ccode);

/*
     Call directory service functions here.
*/
```

```
     NWFreeUnicodeTables( );
}
```

3.2 Example: NWGetLocalToUnicodeHandle

```
/* exul2u.c  -  NWGetLocalToUnicodeHandle / NWLocalToUnicode example
*/
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
#include <unicode.h>
#include <string.h>
#include <assert.h>
void main()
{
   LCONV lconv;
   nint err;
   nptr h = NULL;
   nuint8 src[5] = {'a', 'b', 'c', 'd', 0};   /* Local input string */
   unicode dest[5];
   nuint8 noMap = 0;     /* Use the default replacement char. */
   nuint len;

   /* Get the country ID and code page from the operating system. */
   NWLlocaleconv(&lconv);
   err = NWInitUnicodeTables(lconv.country_id, lconv.code_page);
   assert (err == 0);
   err = NWGetLocalToUnicodeHandle(&h);
   assert (err == 0);
   err = NWLocalToUnicode(h, dest, sizeof(dest), src, noMap, &len);
   assert (err == 0);
   assert (unicmp(dest, L"abcd") == 0); /* Unicode output string */
   assert (len == 5);                   /* Output size includes null */
   NWFreeUnicodeTables();
}
```

3.3 Example: NWGetUnicodeToCollationHandle

```
/* exucoll.c  -  NWGetUnicodeToCollationHandle / NWUnicodeToCollation
*/
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
#include <unicode.h>
#include <assert.h>
void main()
{
   LCONV lconv;
   nint err;
   nptr h = NULL;
   unicode src[5] = {'A', 'a', '1', '*', 0}; /* Unicode input string */
   unicode dest[5];
   unicode noMap = 0;   /* Use the default replacement char. */
```

```
   nuint len;

/* On Win32, it calls wcsxfrm and converts to the same character value.
   The return value does not include the null terminator.

   On other platforms, it uses the Novell collation tables,
     which converts to:
      - the uppercase value of the character for letters
      - zero for NULL
      - 0xFFFF for all other characters
     The return value includes the null terminator.
*/

#ifdef WIN32
   unicode cmp[5] = { 'A', 'a', '1', '*', 0 };
   nuint cmplen = 4;
#else
   unicode cmp[5] = { 'A', 'A', 0xFFFF, 0xFFFF, 0 };
   nuint cmplen = 5;
#endif

   /* Get the country ID and code page from the operating system. */
   NWLlocaleconv(&lconv);
   err = NWInitUnicodeTables(lconv.country_id, lconv.code_page);
   assert (err == 0);
   err = NWGetCollationHandle(&h);
   assert (err == 0);
   err = NWUnicodeToCollation(h, dest, sizeof(dest)/sizeof(unicode),
                              src, noMap, &len);
   assert (err == 0);
   assert (unicmp(dest, cmp) == 0);   /* Unicode output string */
   assert (len == cmplen);
   NWFreeUnicodeTables();
}
```

3.4 Example: NWGetUnicodeToLocalHandle

```
/* exuu2l.c  -  NWGetUnicodeToLocalHandle / NWUnicodeToLocal example
*/
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
#include <unicode.h>
#include <string.h>
#include <assert.h>
void main()
{
   LCONV lconv;
   nint err;
   nptr h = NULL;
   unicode src[5] = {'a', 'b', 'c', 'd', 0}; /* Unicode input string */
   nuint8 dest[5];
   nuint8 noMap = 0;     /* Use the default replacement char. */
```

```
    nuint len;

    /* Get the country ID and code page from the operating system. */
    NWLlocaleconv(&lconv);
    err = NWInitUnicodeTables(lconv.country_id, lconv.code_page);
    assert (err == 0);
    err = NWGetUnicodeToLocalHandle(&h);
    assert (err == 0);
    err = NWUnicodeToLocal(h, dest, sizeof(dest), src, noMap, &len);
    assert (err == 0);
    assert (strcmp(dest, "abcd") == 0);  /* Local output string */
    assert (len == 5);                   /* Output size includes null */
    NWFreeUnicodeTables();
}
```

3.5 Example: NWGetUnicodeToMonocaseHandle

```
/* exumono.c  -  NWGetUnicodeToMonocaseHandle / NWUnicodeToMonocase */
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
#include <unicode.h>
#include <string.h>
#include <assert.h>
void main()
{
    LCONV lconv;
    nint err;
    nptr h = NULL;
    unicode src[5] = {'A', 'B', 'c', 'd', 0}; /* Unicode input string */
    unicode dest[5];
    nuint len;

    /* Get the country ID and code page from the operating system. */
    NWLlocaleconv(&lconv);
    err = NWInitUnicodeTables(lconv.country_id, lconv.code_page);
    assert (err == 0);
    err = NWGetMonocaseHandle(&h);
    assert (err == 0);
    err = NWUnicodeToMonocase(h, dest, sizeof(dest)/sizeof(unicode),
                              src, &len);
    assert (err == 0);
    assert (unicmp(dest, L"abcd") == 0);   /* Unicode output string */
    assert (len == 5);                   /* Output size includes null */
    NWFreeUnicodeTables();
}
```

3.6 Example: NWInitUnicodeTables

```
/* exuinit.c  -  NWInitUnicodeTables / NWFreeUnicodeTables example */
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
```

```
#include <unicode.h>
#include <assert.h>
void main()
{
   LCONV lconv;
   nint err;

   /* Get the country ID and code page from the operating system. */
   NWLlocaleconv(&lconv);
   err = NWInitUnicodeTables(lconv.country_id, lconv.code_page);

   /* Be sure to check return value before proceeding. */
   assert (err == 0);

   /*
      .
      .
      .
   */
   NWFreeUnicodeTables();
}
```

3.7 Example: NWLocalToUnicode with Buffer Overflow

```
/* exul2u_1.c  -  NWLocalToUnicode with buffer overflow example */
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
#include <unicode.h>
#include <string.h>
#include <assert.h>
void main()
{
   LCONV lconv;
   nint err;
   nptr h = NULL;
   nuint8 src[5] = {'a', 'b', 'c', 'd', 0};   /* Local input string */
   unicode dest[3];    /* This size is too small to hold the result! */
   nuint8 noMap = 0;   /* Use the default replacement char. */
   nuint len;

   /* Get the country ID and code page from the operating system. */
   NWLlocaleconv(&lconv);
   err = NWInitUnicodeTables(lconv.country_id, lconv.code_page);
   assert (err == 0);
   err = NWGetLocalToUnicodeHandle(&h);
   assert (err == 0);
   err = NWLocalToUnicode(h, dest, sizeof(dest)/sizeof(unicode), src,
                          noMap, &len);
   assert (err == 0);

/*   Win32 clients call Windows MultiByteToWideChar.
```

```
     If buffer overflows, it does not terminate the output string,
     and returns zero length.
     It ignores noMap and uses the windows default replacement char.
*/

#ifdef WIN32
   assert (unincmp(dest, L"abc", 3) == 0);
   assert (len == 0);

/*   Other platforms terminate the string and return the
     number of chars written. */

#else
   assert (unicmp(dest, L"ab") == 0);
   assert (len == 3);           /* 2 uni chars and a null were written
*/
#endif
   NWFreeUnicodeTables();
}
```

3.8 Example: NWUnicodeCompare

```
/* exucomp.c  -  NWUnicodeCompare example.
   Compare 2 unicode characters in collation sequence.
*/
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
#include <unicode.h>
#include <assert.h>
void main()
{
   LCONV lconv;
   nint ret;
   nptr h = NULL;

   /* Get the country ID and code page from the operating system. */
   NWLlocaleconv(&lconv);
   ret = NWInitUnicodeTables(lconv.country_id, lconv.code_page);
   assert (ret == 0);
   ret = NWGetCollationHandle(&h);
   assert (ret == 0);
   ret = NWUnicodeCompare(h, 'A', 'B');
   assert (ret == -1);                    /* A < B */

/* On WNT, it calls StringCompareW, which considers a < A.
   The collation order is  ...aA..bB..cC...
*/

#ifdef N_PLAT_WNT
   ret = NWUnicodeCompare(h, 'A', 'a');
   assert (ret == 1);                     /* A > a */
   ret = NWUnicodeCompare(h, 'a', 'B');
```

```
   assert (ret == -1);
   ret = NWUnicodeCompare(h, 'B', 'a');
   assert (ret == 1);

/* On W95, it calls wcsxfrm, which considers A < a.
   The collation order is the same as the character value:
      ...ABC..Z..abc..z...
*/

#elif defined WIN32
   ret = NWUnicodeCompare(h, 'A', 'a');
   assert (ret == -1);                    /* A < a */
   ret = NWUnicodeCompare(h, 'a', 'B');
   assert (ret == 1);
   ret = NWUnicodeCompare(h, 'B', 'a');
   assert (ret ==-1);

/* On NLM it uses the Novell collation tables.
   Upper and lowercase letters are considered equal.
*/

#else
   ret = NWUnicodeCompare(h, 'A', 'a');
   assert (ret == 0);                    /* A < a */
   ret = NWUnicodeCompare(h, 'a', 'B');
   assert (ret == -1);
   ret = NWUnicodeCompare(h, 'B', 'a');
   assert (ret ==1);
#endif
   NWFreeUnicodeTables();
}
```

3.9 Example: NWUnicodeToLocal with Buffer Overflow

```
/* exuu2l_1.c  -  NWUnicodeToLocal with buffer overflow example */
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
#include <unicode.h>
#include <string.h>
#include <assert.h>
void main()
{
   LCONV lconv;
   nint err;
   nptr h = NULL;
   unicode src[5] = {'a', 'b', 'c', 'd', 0}; /* Unicode input string */
   nuint8 dest[3];     /* This size is too small to hold the result! */
   nuint8 noMap = 0;   /* Use the default replacement char. */
   nuint len;

   /* Get the country ID and code page from the operating system. */
```

```
   NWLlocaleconv(&lconv);
   err = NWInitUnicodeTables(lconv.country_id, lconv.code_page);
   assert (err == 0);
   err = NWGetUnicodeToLocalHandle(&h);
   assert (err == 0);
   err = NWUnicodeToLocal(h, dest, sizeof(dest), src, noMap, &len);
   assert (err == 0);

/*   Win32 clients call Windows WideCharToMultiByte.
     If buffer overflows, it does not terminate the output string,
     and returns zero length.
     It ignores noMap and uses the windows default replacement char.
*/

#ifdef WIN32
   assert (strncmp(dest, "abc", 3) == 0);
   assert (len == 0);

/* Other platforms terminate the string and return the
   number of chars written.
*/

#else
   assert (strcmp(dest, "ab") == 0);
   assert (len == 3);            /* 2 bytes and a null were written */
#endif
   NWFreeUnicodeTables();
}
```

3.10 Example: NWUnicodeToLocal with Unmappable Character

```
/* exuu2l_2.c  -  NWUnicodeToLocal example with unmappable character */
#define NWL_EXCLUDE_TIME
#define NWL_EXCLUDE_FILE
#include <nwlocale.h>
#include <unicode.h>
#include <string.h>
#include <assert.h>
void main()
{
   LCONV lconv;
   nint err;
   nptr h = NULL;
   unicode src[5] = {'a', 0xFFFD, 'c', 'd', 0};   /* Unmappable char */
   nuint8 dest[5];
   nuint8 noMap = 0;     /* Use the default replacement char. */
   nuint len;

/* Win32 platforms ignore noMap and use the Windows default char ('?').
   Non-win32 platforms use the given noMap character, or if it is zero,
   uses the default nomap character from the Novell rules table,
   which for US English, is 0xFD on NLM
```

```
*/

#ifdef WIN32
   nuint8 expect[5]  = {'a', '?', 'c', 'd', 0};   /* Win32 output */
#elif defined NWWIN
   nuint8 expect[5]  = {'a', 0x03, 'c', 'd', 0};  /* NLM output */
#endif

   /* Get the country ID and code page from the operating system. */
   NWLlocaleconv(&lconv);
   err = NWInitUnicodeTables(lconv.country_id, lconv.code_page);
   assert (err == 0);
   err = NWGetUnicodeToLocalHandle(&h);
   assert (err == 0);
   err = NWUnicodeToLocal(h, dest, sizeof(dest), src, noMap, &len);
   assert (err == 0);
   assert (strcmp(dest, expect) == 0);  /* Local output string */
   assert (len == 5);                   /* Output size includes null */
   NWFreeUnicodeTables();
}
```

3.11 Example: unicat

```
/*  excat.c - Sample code for unicat function */
#include <nunicode.h>
#include <assert.h>
void main(void)
{
   unicode str[10];
   punicode ret;
   unicpy(str, L"Star");
   ret = unicat(str, L"fish"); /* Append "fish" onto the end of str */
   assert(unicmp(str, L"Starfish") == 0);
   assert(ret==str);
}
```

3.12 Example: unichr

```
/*  exchr.c - unichr sample.  Find the first occurrence of a character.
*/
#include <nunicode.h>
#include <assert.h>
void main(void)
{
   punicode puni = L"banana";
   punicode ptr;
   ptr = unichr(puni, 'a');       /* Returns ptr to the first 'a' */
   assert(unicmp(ptr, L"anana") == 0);
   ptr = unichr(puni, 'Z');       /* If not found, return NULL. */
   assert (ptr == NULL);
}
```

3.13 Example: unicmp

```
/*  excmp.c - Sample code for unicmp.  Case sensitive comparison. */

#include <nunicode.h>
#include <assert.h>
void main(void)
{
  nint result;

  result = unicmp(L"Cat", L"Cat");   /* zero - strings match exactly */
  assert (result == 0);
  result = unicmp(L"cat", L"catch"); /* non-zero - no match */
  assert (result != 0);
  result = unicmp(L"CAT", L"cat");   /* Case-sensitive.  No match */
  assert (result != 0);
  result = unicmp(L"cat", L"dog");   /* Negative - cat < dog  */
  assert (result < 0);
  result = unicmp(L"dog", L"cat");   /* Positive - dog > cat  */
  assert (result > 0);
}
```

3.14 Example: unicpy

```
/*  excpy.c - Sample code for unicpy function.
A bug in earlier versions of the library caused unicpy to return a
pointer to the end of the destination string instead of the beginning.
*/

#include <nunicode.h>
#include <assert.h>
void main(void)
{
   unicode str[11];
   punicode ret;
   ret = unicpy(str, L"New string");
   assert(unicmp(str, L"New string") == 0);
   assert(ret==str);
}
```

3.15 Example: unicspn

```
/*  excspn.c - Sample code for unicspn.

    Return initial length of first string not containing any
    characters of the second string.
*/

#include <nunicode.h>
#include <assert.h>
void main(void)
{
  nint result;
```

```
  result = unicspn(L"testfile.dat", L".");
  assert (result == 8);
  result = unicspn(L"server\\vol:file.ext", L".:\\");
  assert (result == 6);
}
```

3.16 Example: uniicmp

```
/*  exicmp.c - Sample code for uniicmp.  Case insensitive comparison.
*/

# include <nunicode.h>
#include <assert.h>
void main(void)
{
  nint result;
  result = uniicmp(L"CAT", L"cat");     /* zero - strings match */
  assert (result == 0);
  result = uniicmp(L"cat", L"catch");   /* non-zero - no match */
  assert (result != 0);
  result = uniicmp(L"cat", L"dog");     /* Negative - cat < dog  */
  assert (result < 0);
  result = uniicmp(L"dog", L"cat");     /* Positive - dog > cat  */
  assert (result > 0);
}
```

3.17 Example: unilen

```
/*  exlen.c - Sample code for unilen function.
    Returns the length of the string in unicode characters,
    excluding the terminating NULL.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
   nint i;
   i = unilen(L"cat");
   assert (i==3);
   i = unilen(L"kitten");
   assert (i==6);
}
```

3.18 Example: unincat

```
/*  exncat.c - Sample code for unincat.
    Append n characters of second string onto first string.
*/
```

```
#include <nunicode.h>
#include <assert.h>

void main(void)
{
  unicode  s1[20];
  punicode ret;

  unicpy(s1, L"Star");

  ret = unincat(s1, L"Fisherman", 4);       /* Result is "StarFish" */

  assert (unicmp(ret, L"StarFish") == 0);
  assert (ret == s1);
}
```

3.19 Example: unincmp

```
/*  exncmp.c - Sample code for unincmp.
    Case sensitive comparison of n chars.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
  nint result;

  result = unincmp(L"cat", L"catch", 3);   /* Match in first 3 chars */
  assert (result == 0);
  result = unincmp(L"cat", L"catch", 4);   /* Not a match in 4 chars */
  assert (result != 0);

  result = unincmp(L"CAT", L"cat", 2);  /* Case-sensitive.  No match */
  assert (result != 0);
}
```

3.20 Example: uninicmp

```
/*  exnicmp.c - Sample code for uninicmp.  Case insensitive length
comparison.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
  nint result;
  result = uninicmp(L"CAT", L"cat", 3);  /* Case-insensitive.  Match */
  assert (result == 0);
  result = uninicmp(L"Cat", L"catch", 3); /* Match in first 3 chars */
```

```
  assert (result == 0);
  result = uninicmp(L"cat", L"catch", 4);  /* Not a match in 4 chars */
  assert (result != 0);
}
```

3.21 Example: uninset

```
/*  exnset.c - Sample code for uninset.
    Fill up to n characters of existing string.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
  unicode  s1[20];
  punicode ret;
  unicode  fill = '.';

  unicpy(s1, L"StarFish");

  ret = uninset(s1, fill, 4);

  assert (unicmp(s1, L"....Fish") == 0);
  assert (ret == s1);

/* Fill stops after 8 chars when it hits a NULL in the destination. */
  ret = uninset(s1, fill, 20);
  assert (unicmp(s1, L"........") == 0);
}
```

3.22 Example: unipcpy

```
/*  expcpy.c - Sample code for unipcpy function.
Copy string and return pointer to the end of the destination string
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
   unicode str[50];
   punicode p ;
   p = unipcpy(str, L"xx");
   p = unipcpy(p, L"yy");
   p = unipcpy(p, L"zz");
   assert(unicmp(str, L"xxyyzz") == 0);
   assert( *p == 0);  /* Return value points to the terminating null */
}
```

3.23 Example: unipbrk

```
/*  expbrk.c - Sample code for unipbrk.

    Returns a pointer to the first occurrence in s1
    of any character in s2.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
  punicode ret;

  /* Search for the first dot, colon, or backslash character.  */
  ret = unipbrk(L"server\\vol:file.ext", L".:\\");

  /* ret points to the backslash character. */
  assert (*ret == '\\');
}
```

3.24 Example: unirchr

```
/*  exrchr.c - unirchr sample.  Find the last occurrence of a
character.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
   punicode puni = L"bananas";
   punicode ptr;
   ptr = unirchr(puni, 'a');         /* Returns ptr to the last 'a' */
   assert(unicmp(ptr, L"as") == 0);
   ptr = unichr(puni, 'Z');         /* If not found, return NULL. */
   assert (ptr == NULL);
}
```

3.25 Example: unirev

```
/*  exrev.c - Sample code for unirev function.
    Reverse string in place.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
   unicode str[5];
```

```
   punicode ret;
   unicpy(str,L"abcd");
   ret = unirev(str);
   assert(unicmp(str, L"dcba") == 0);
   assert(ret==str);
}
```

3.26 Example: uniset

```
/*  exset.c - Sample code for uniset.
    Replace all characters in a string.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
  unicode  s1[20];
  punicode ret;
  unicode  fill = '.';

  unicpy(s1, L"abcd");

  /* Fill stops when it hits a NULL in the destination. */
  ret = uniset(s1, fill);
  assert (unicmp(s1, L"....") == 0);
  assert (ret == s1);
}
```

3.27 Example: unisize

```
/*  exsize.c - Sample code for unisize function.
    Returns the length of the string in bytes,
    including the terminating NULL.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
   nint i;
   i = unisize(L"cat");
   assert (i==8);
   i = unisize(L"");
   assert (i==2);
}
```

3.28 Example: unispn

```
/*  exspn.c - Sample code for unispn.

    Return initial length of first string containing only
    characters of the second string.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
  nint result;

  result = unispn(L"abcabcxx", L"bac");
  assert (result == 6);
}
```

3.29 Example: unistr

```
/*  exstr.c - Sample code for unistr.

   Search for a substring of a string.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
  punicode ret;

  ret = unistr(L"aa bb cc dd", L"cc");
  assert (unicmp(ret, L"cc dd") == 0);
  ret = unistr(L"aa bb cc dd", L"xx");   /* "xx" does not occur */
  assert (ret == NULL);
}
```

3.30 Example: unitok

```
/*  extok.c - Sample code for unitok.

    Returns next component of string as defined by tokens.
*/

#include <nunicode.h>
#include <assert.h>

void main(void)
{
  unicode str[30];
  unicode tokens[4] = { '.', ':', '\\', 0 };
```

```
  punicode ret;
  unicpy(str, L"server\\vol:file.ext");
  ret = unitok(str, tokens);
  assert (unicmp(ret, L"server") == 0);

  ret = unitok(NULL, tokens);
  assert (unicmp(ret, L"vol") == 0);
  ret = unitok(NULL, tokens);
  assert (unicmp(ret, L"file") == 0);
  ret = unitok(NULL, tokens);
  assert (unicmp(ret, L"ext") == 0);
  ret = unitok(NULL, tokens);
  assert (ret == NULL);
}
```

Functions

4

This documentation alphabetically lists the Unicode functions and describes their purpose, syntax, parameters, and return values.

NWFreeUnicodeTables

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

Function NWFreeUnicodeTables
  : nint;
```

Return Values

0x0000	Successful is always returned

Remarks

Example: NWFreeUnicodeTables

See Section 3.6, “Example: NWInitUnicodeTables,” on page 46

See Also

NWInitUnicodeTables (page 71)

NWGetCollationHandle

Sets a handle to the Unicode-to-collation rule table

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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

Parameters

handle

(OUT) Points to the rule handle.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD

Remarks

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

See Also

NWInitUnicodeTables (page 71), NWLoadRuleTable (page 73), NWUnicodeCompare (page 85)

NWGetLocalToUnicodeHandle

Sets rule handle to the local-to-Unicode rule table

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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

Parameters

handle

(OUT) Points to the rule handle which should be set to NULL before you call this function.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD

Remarks

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

See Also

NWInitUnicodeTables (page 71), NWLoadRuleTable (page 73), NWLocalToUnicode (page 75)

NWGetMonocaseHandle

Sets a rule handle to the Unicode-to-monocase rule table

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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

Parameters

handle

(OUT) Points to the rule handle which should be set to NULL before you call this function.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD

Remarks

If the rule table has not been loaded, the pointer is set to NULL. See Section 3.5, “Example: NWGetUnicodeToMonocaseHandle,” on page 46.

See Also

NWInitUnicodeTables (page 71), NWLoadRuleTable (page 73), NWUnicodeToMonocase (page 92)

NWGetUnicodeToLocalHandle

Sets a rule handle to the Unicode-to-local rule table

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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

Parameters

handle

(OUT) Points to the rule handle which should be set to NULL before you call this function.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD

Remarks

If the rule table has not been loaded, the pointer is set to NULL. See Section 3.4, “Example: NWGetUnicodeToLocalHandle,” on page 45.

See Also

NWInitUnicodeTables (page 71), NWLoadRuleTable (page 73), NWUnicodeToLocal (page 89)

NWInitUnicodeTables

Sets up and initializes four conversion tables necessary to support Unicode

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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 (*Return Values for C*) for more information.

0x0000	Successful
0xFE11	UNI_LOAD_FAILED
0xFE12	UNI_NO_MEMORY
0xFE13	UNI_NO_PERMISSION
0xFE14	UNI_TOO_MANY_FILES

0xFE15	UNI_NO_SUCH_FILE
0xFE16	UNI_FUTURE_OPCODE
0xFE0B	UNI_OPEN_FAILED
0xFE0E	UNI_RULES_CORRUPT

Remarks

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

NWInitUnicodeTables or NWLoadRuleTable must be successfully called before these functions can be called:

NWGetUnicodeToLocalHandle
NWGetLocalToUnicodeHandle
NWGetMonocaseHandle
NWGetCollationHandle

See Section 3.6, “Example: NWInitUnicodeTables,” on page 46.

See Also

NWFreeUnicodeTables (page 64), NWLlocaleconv (Internationalization), NWLoadRuleTable (page 73)

NWLoadRuleTable

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

NetWare Server: 4.x, 5.x, 6.x

Platform: Windows 95, Windows 98, Windows NT

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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 (*Return Values for C*) for more information.

0x0000	Successful
0xFE11	UNI_LOAD_FAILED
0xFE12	UNI_NO_MEMORY
0xFE13	UNI_NO_PERMISSION
0xFE14	UNI_TOO_MANY_FILES
0xFE15	UNI_NO_SUCH_FILE
0xFE16	UNI_FUTURE_OPCODE

0xFE0B	UNI_OPEN_FAILED
0xFE0E	UNI_RULES_CORRUPT

Remarks

NWLoadRuleTable replaces NWInitUnicodeTables and the following calls:

NWGetUnicodeToLocalHandle
NWGetLocalToUnicodeHandle
NWGetMonocaseHandle
NWGetCollationHandle

Win32 platforms do not use the Novell unicode tables and just return success. NLM platforms do not support these routines.

For information about rule tables, see NWInitUnicodeTables (page 71).

See Also

NWInitUnicodeTables (page 71), NWUnloadRuleTable (page 94)

NWLocalToUnicode

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) NWLocalToUnicode (
   nptr               ruleHandle,
   punicode           dest,
   nuint32            maxLen,
   const void N_FAR  *src,
   unicode            noMap,
   pnuint             len,
   nuint32            allowNoMapFlag);
```

Pascal Syntax

```
uses netwin32

Function NWLocalToUnicode
  (ruleHandle : nptr;
   dest : punicode;
   maxLen : nuint32;
   src : nptr;
   noMap : unicode;
   len : pnuint;
   allowNoMapFlag : nuint32 (*optional, available on NLM only*) );
```

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).

`allowNoMapFlag`

(IN) (Optional, available only on NLM) Specifies whether to replace an unmappable character or simply return an error. (See explanation in "Remarks" below.)

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD
0xFE0D	UNI_NO_DEFAULT
0xFE0E	UNI_RULES_CORRUPT
0xFE0F	UNI_HANDLE_MISMATCH

Remarks

For cross-platform operations, NWLocalToUnicode uses only the first six parameters. The seventh parameter is optionally available only for NLM applications.

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.

The `allowNoMapFlag` parameter provides another optional behavior, but only for the NLM platform (N_PLAT_NLM must be defined). By default, on NLM applications NWLocalToUnicode handles unmappable characters in the same way as non-Win32 clients described above. To specify behavior explicitly, define EXCLUDE_UNICODE_NLM_COMPATIBILITY_MACROS and pass one of the following values for allowNoMapFlag:

- DONT_USE_NOMAP_CHAR- NWLocalToUnicode returns UNI_NO_DEFAULT any time an unmappable character is encountered.
- USE_NOMAP_CHAR-NWLocalToUnicode provides the default behavior of non-Win32 clients described above.

For examples, see

- Section 3.2, “Example: NWGetLocalToUnicodeHandle,” on page 44
- Section 3.7, “Example: NWLocalToUnicode with Buffer Overflow,” on page 47

See Also

NWGetLocalToUnicodeHandle (page 66), NWUnicodeToLocal (page 89)

NWLUnicodeToUTF8

Converts a Unicode string to UTF-8 format.

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
N_GLOBAL_LIBRARY(nint) NWLUnicodeToUTF8 (
   punicode   uniStr,
   nuint      maxSize,
   pnuint8    utf8Str,
   pnuint     utf8Size);
```

Pascal Syntax

```
uses netwin32

Function NWLUnicodeToUTF8
  (uniStr : punicode;
   maxSize : nuint;
   utf8Str : pnuint8;
   utf8Size : pnuint
): nint;
```

Parameters

`uniStr`

(IN) Points to the Unicode-formatted string.

`maxSize`

(IN) Specifies the number of bytes allocated for `utf8Str`.

`utf8Str`

(OUT) Points to a buffer that holds the new UTF-8 string.

`utf8Size`

(OUT) Points to the length (in bytes) of the string contained in `utf8Str`, including the string termination character (optional).

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0	N_SUCCESS
-500	UNI_INSUFFICIENT_BUFFER

Remarks

For sample code, see utf8.c (../../../samplecode/clib_sample/intl_utf8/utf8.c.html).

See Also

NWLUnicodeToUTF8Size (page 80), NWLUTF8ToUnicode (page 81), NWLUTF8ToUnicodeSize (page 83)

NWLUnicodeToUTF8Size

Computes the number of bytes that must be allocated to store a Unicode string in UTF-8 format.

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
N_GLOBAL_(nuint) NWLUnicodeToUTF8Size (
   punicode   uniStr);
```

Pascal Syntax

```
uses netwin32

Function NWLUnicodeToUTF8Size
  (uniStr : punicode
): nint;
```

Parameters

uniStr

(IN) Points to a Unicode string.

Return Values

Returns the size (in bytes) required to store a Unicode string (including the string termination character) in UTF-8 format.

Remarks

For sample code, see utf8.c (../../../samplecode/clib_sample/intl_utf8/utf8.c.html).

See Also

NWLUnicodeToUTF8 (page 78), NWLUTF8ToUnicode (page 81), NWLUTF8ToUnicodeSize (page 83)

NWLUTF8ToUnicode

Converts a UTF-8 string to Unicode.

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
N_GLOBAL_LIBRARY(nint) NWLUTF8ToUnicode (
   pnuint8    utf8Str,
   nuint      maxSize,
   punicode   uniStr,
   pnuint     uniSize,
   ppnstr     badSequence);
```

Pascal Syntax

```
uses netwin32

Function NWLUTF8ToUnicode
  (utf8Str : pnuint8;
  maxSize : nuint;
  uniStr : punicode;
  uniSize : pnuint;
  badSequence : ppnstr
): nint;
```

Parameters

utf8Str

(IN) Points to the UTF-8 string to be converted.

maxSize

(IN) Specifies the size (in bytes) allocated for `uniStr`.

uniStr

(OUT) Points to a buffer to hold the converted Unicode string.

uniSize

(OUT) Points to the length (in bytes) of the converted string contained in `uniStr`, including the string termination character (optional).

badSequence

(OUT) Points to a bad UTF-8 sequence if an error occurs (optional).

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0	N_SUCCESS
-500	UNI_INSUFFICIENT_BUFFER
-506	UNI_ILLEGAL_UTF8_CHARACTER

Remarks

For sample code, see utf8.c (../../../samplecode/clib_sample/intl_utf8/utf8.c.html).

See Also

NWLUnicodeToUTF8 (page 78), NWLUnicodeToUTF8Size (page 80), NWLUTF8ToUnicodeSize (page 83)

NWLUTF8ToUnicodeSize

Computes the number of bytes that must be allocated to convert a UTF-8 string to Unicode.

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
N_GLOBAL_LIBRARY(nint) NWLUTF8ToUnicodeSize (
   pnuint8   utf8Str,
   pnuint    size);
```

Pascal Syntax

```
uses netwin32

Function NWLUTF8ToUnicodeSize
  (utf8Str : pnuint8;
  size : pnuint
): nint;
```

Parameters

utf8Str

(IN) Points to a UTF-8 string.

size

(OUT) Points to the number of bytes necessary to represent the Unicode string, including the string termination character.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0	N_SUCCESS
-506	UNI_ILLEGAL_UTF8_CHARACTER

Remarks

For sample code, see utf8.c (../../../samplecode/clib_sample/intl_utf8/utf8.c.html).

See Also

NWLUnicodeToUTF8 (page 78), NWLUnicodeToUTF8Size (page 80), NWLUTF8ToUnicode (page 81)

NWUnicodeCompare

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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
>0	If chr1 > chr2

Remarks

For sample code, see Section 3.8, “Example: NWUnicodeCompare,” on page 48.

See Also

NWGetCollationHandle (page 65), NWUnicodeToCollation (page 87)

NWUnicodeToCollation

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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 (*Return Values for C*) for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD
0xFE0D	UNI_NO_DEFAULT
0xFE0E	UNI_RULES_CORRUPT
0xFE0F	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.

For sample code, see Section 3.3, “Example: NWGetUnicodeToCollationHandle,” on page 44.

See Also

NWGetCollationHandle (page 65), NWUnicodeCompare (page 85)

NWUnicodeToLocal

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY (nint) NWUnicodeToLocal (
   nptr                    ruleHandle,
   pnuint8                 dest,
   size_t                  maxLen,
   const unicode N_FAR    *src,
   nuint8                  noMap,
   size_t N_FAR           *len,
   nuint32                 allowNoMapFlag);
```

Pascal Syntax

```
uses netwin32

Function NWUnicodeToLocal
  (ruleHandle : nptr;
   dest : nptr;
   maxLen : nuint32;
   src : punicode;
   noMap : nuint8;
   len : pnuint;
   allowNoMapFlag : nuint32 (*optional, available on NLM only*) );
```

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).

allowNoMapFlag

(IN) (Optional, available only on NLM) Specifies whether to replace an unmappable character or simply return an error. (See explanation in "Remarks" below.)

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD
0xFE0D	UNI_NO_DEFAULT
0xFE0E	UNI_RULES_CORRUPT
0xFE0F	UNI_HANDLE_MISMATCH

Remarks

For cross-platform operations, NWUnicodeToLocal uses only the first six parameters. The seventh parameter is optionally available only for NLM applications.

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.

The `allowNoMapFlag` parameter provides another optional behavior, but only for the NLM platform (N_PLAT_NLM must be defined). By default, on NLM applications NWUnicodeToLocal handles unmappable characters in the same way as non-Win32 clients described above. To specify behavior explicitly, define EXCLUDE_UNICODE_NLM_COMPATIBILITY_MACROS and pass one of the following values for allowNoMapFlag:

- DONT_USE_NOMAP_CHAR- NWUnicodeToLocal returns UNI_NO_DEFAULT any time an unmappable character is encountered.
- USE_NOMAP_CHAR-NWUnicodeToLocal provides the default behavior of non-Win32 clients described above.

For sample code, see

- Section 3.4, “Example: NWGetUnicodeToLocalHandle,” on page 45
- Section 3.9, “Example: NWUnicodeToLocal with Buffer Overflow,” on page 49
- Section 3.10, “Example: NWUnicodeToLocal with Unmappable Character,” on page 50

See Also

NWGetUnicodeToLocalHandle (page 69), NWLocalToUnicode (page 75)

NWUnicodeToMonocase

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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 including the NULL terminator.

Return Values

These are common return values; see Return Values (*Return Values for C*) 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.

For sample code, see Section 3.5, “Example: NWGetUnicodeToMonocaseHandle,” on page 46.

See Also

NWGetMonocaseHandle (page 67)

NWUnloadRuleTable

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

NetWare Server: 4.x, 5.x, 6.x

Platform: Windows 95, Windows 98, Windows NT

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

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

Pascal Syntax

```
uses netwin32

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

Parameters

ruleHandle

(IN) Points to the rule table handle.

Return Values

These are common return values; see Return Values (*Return Values for C*) for more information.

0x0000	Successful
0xFE10	UNI_HANDLE_BAD

See Also

NWLoadRuleTable (page 73)

NWUSByteToUnicode

Converts a NULL-terminated byte string into a Unicode string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSByteToUnicode (
   punicode              unicodeOutput,
   nuint                 outputBufferLen,
   const nuint8 N_FAR   *byteInput,
   pnuint                actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSByteToUnicode
  (unicodeOutput : punicode;
  outputBufferLen : nuint;
  byteInput : const nuint8;
  actualLength : pnuint
): nint;
```

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 esbuni.c (../../../samplecode/clib_sample/intl_esbuni/esbuni.c.html) for sample code.

See Also

NWUSByteToUnicodePath (page 97)

NWUSByteToUnicodePath

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSByteToUnicodePath (
   punicode              unicodeOutput,
   nuint                 outputBufferLen,
   const nuint8 N_FAR   *byteInput,
   pnuint                actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSByteToUnicodePath
  (unicodeOutput : punicode;
  outputBufferLen : nuint;
  byteInput : const nuint8;
  actualLength : pnuint
): nint;
```

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 esbunip.c (../../../samplecode/clib_sample/intl_esbunip/esbunip.c.html) for sample code.

See Also

NWUSByteToUnicode (page 95)

NWUSGetCodePage

Returns the codepage and country of the standard converter

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

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

Pascal Syntax

```
uses netwin32

Function NWUSGetCodePage
  (pCodePage : pnuint;
  pCountry : pnuint
): nint;
```

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 returns the codepage of the standard byte/Unicode converter. It also returns the country ID of the system. If NWUSStandardUnicodeInit has not yet been called, NWUSGetCodePage returns the codepage and country ID that will be used when NWUSStandardUnicodeInit is called.

If an error occurs when NWUSStandardUnicodeInit is called, NWUSGetCodePage might be useful for printing an error message to tell the user which converters are needed.

See esgetcp.c (../../../samplecode/clib_sample/intl_esgetcp/esgetcp.c.html) for sample code.

See Also

NWUSStandardUnicodeInit (page 105)

NWUSLenByteToUnicode

Converts a length-specified byte string into a Unicode string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSLenByteToUnicode (
   punicode              unicodeOutput,
   nuint                 outputBufferLen,
   const nuint8 N_FAR   *byteInput,
   nuint                 inLength,
   pnuint                actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSLenByteToUnicode
  (unicodeOutput : punicode;
  outputBufferLen : nuint;
  byteInput : const nuint8;
  inLength : nuint;
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	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, see “Length-Specified Byte String Conversion” on page 25.

See eslbuni.c (../../../samplecode/clib_sample/intl_eslbuni/eslbuni.c.html) for sample code.

See Also

NWUSLenByteToUnicodePath (page 103), NWUSUnicodeToUntermByte (page 116)

NWUSLenByteToUnicodePath

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSLenByteToUnicodePath (
   punicode              unicodeOutput,
   nuint                 outputBufferLen,
   const nuint8 N_FAR   *byteInput,
   nuint                 inLength,
   pnuint                actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSLenByteToUnicodePath
  (unicodeOutput : punicode;
  outputBufferLen : nuint;
  byteInput : const nuint8;
  inLength : nuint;
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	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, see “Length-Specified Byte String Conversion” on page 25.

See eslbunip.c (../../../samplecode/clib_sample/intl_eslbunip/eslbunip.c.html) for sample code.

See Also

NWUSLenByteToUnicode (page 101), NWUSUnicodeToUntermBytePath (page 118)

NWUSStandardUnicodeInit

Loads both the lower and upper case converters needed for the standard Unicode functions

Local Servers: blocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSStandardUnicodeInit  (
   void);
```

Pascal Syntax

```
uses netwin32

Function NWUSStandardUnicodeInit
  (void
): nint;
```

Return Values

0x0000	SUCCESSFUL
0xFDE8	NWU_CONVERTER_CORRUPT
0xFDE6	NWU_OUT_OF_MEMORY
0xFDE5	NWU_READ_FAILED (NIOS only)
0xFDE4	NWU_OPEN_FAILED (NIOS only)
0xFDE3	NWU_NO_PERMISSION (NIOS only)
0xFDE2	NWU_TOO_MANY_FILES (NIOS only)
0xFDE1	NWU_CONVERTER_NOT_FOUND

For an explanation of the NWU_CONVERTER_NOT_FOUND error, see "NWU_CONVERTER_NOT_FOUND Error" on page 23

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 esinit.c (../../../samplecode/clib_sample/intl_esinit/esinit.c.html) for sample code.

NOTE: Access to global variables initialized with a standard converter can vary according to platform, as explained in "Conversion Operations" on page 21.

See Also

NWUSStandardUnicodeRelease (page 109)

NWUSStandardUnicodeOverride

Replaces the current standard converter with a converter for the specified codepage

Local Servers: blocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSStandardUnicodeOverride (
   nuint   codepage);
```

Pascal Syntax

```
uses netwin32

Function NWUSStandardUnicodeOverride
  (codepage : nuint
): nint;
```

Parameters

codepage

(IN) Specifies the codepage of the replacement standard converter.

Return Values

0x0000	SUCCESSFUL
0xFDE8	NWU_CONVERTER_CORRUPT
0xFDE6	NWU_OUT_OF_MEMORY
0xFDE5	NWU_READ_FAILED (NIOS only)
0xFDE4	NWU_OPEN_FAILED (NIOS only)
0xFDE3	NWU_NO_PERMISSION (NIOS only)
0xFDE2	NWU_TOO_MANY_FILES (NIOS only)
0xFDE1	NWU_CONVERTER_NOT_FOUND

For an explanation of the NWU_CONVERTER_NOT_FOUND error, see "NWU_CONVERTER_NOT_FOUND Error" on page 23

Remarks

NWUSStandardUnicodeOverride replaces the current standard Unicode converter with the converter specified in the `codepage` parameter. The specified converter performs all standard Unicode conversions until it is changed with another call to NWUSStandardUnicodeOverride or is released with a call to NWUSStandardUnicodeRelease.

If a standard converter has not been loaded with a call to NWUSStandardUnicodeInit, a call to NWUSStandardUnicodeOverride initializes the specified standard converter and sets up global variables needed for standard Unicode conversions.

NWUSStandardUnicodeOverride cannot be nested as can NWUSStandardUnicodeInit. However, NWUSStandardUnicodeOverride can be called multiple times to change standard converters.

When conversion operations are completed, release the current converter and associated resources by calling NWUSStandardUnicodeRelease.

NOTE: Access to global variables initialized with a standard converter can vary according to platform, as explained in "Conversion Operations" on page 21.

See Also

NWUSStandardUnicodeInit (page 105), NWUSStandardUnicodeRelease (page 109)

NWUSStandardUnicodeRelease

Releases all resources allocated by the NWUSStandardUnicodeInit function

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(void) NWUSStandardUnicodeRelease  (
   void);
```

Pascal Syntax

```
uses netwin32

Function NWUSStandardUnicodeRelease
  (void
): void;
```

Remarks

Call the NWUSStandardUnicodeRelease function only after calling the NWUSStandardUnicodeInit function. Otherwise, resources required by another application could be released.

See esrel.c (../../../samplecode/clib_sample/intl_esrel/esrel.c.html) for sample code.

See Also

NWUSStandardUnicodeInit (page 105)

NWUSUnicodeToByte

Converts a NULL-terminated Unicode string into a byte string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToByte (
   pnuint8                byteOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSUnicodeToByte
  (byteOutput : pnuint8;
  outputBufferLen : nuint;
  unicodeInput : punicode;
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	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 number of characters in the input string (or three times longer than the number of bytes in 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 esunib.c (../../../samplecode/clib_sample/intl_esunib/esunib.c.html) for sample code.

NWUSUnicodeToBytePath

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToBytePath (
   pnuint8                byteOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSUnicodeToBytePath
  (byteOutput : nuint8;
  outputBufferLen : nuint;
  unicodeInput : const unicode;
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	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 number of characters in the input string (or three times longer than the number of bytes in 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 esunibp.c (../../../samplecode/clib_sample/intl_esunibp/esunibp.c.html) for sample code.

See Also

NWUSUnicodeToByte (page 110)

NWUSUnicodeToLowerCase

Converts a NULL-terminated Unicode string to Unicode lower case characters

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToLowerCase (
   punicode               lowerCaseOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSUnicodeToLowerCase
  (lowerCaseOutput : punicode;
  outputBufferLen : nuint;
  unicodeInput : const unicode;
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	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 esunilow.c (../../../samplecode/clib_sample/intl_esunilow/esunilow.c.html) for sample code.

NWUSUnicodeToUntermByte

Converts a NULL-terminated Unicode string into an unterminated byte string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToUntermByte  (
   pnuint8                byteOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSUnicodeToUntermByte
  (byteOutput : pnuint8;
  outputBufferLen : nuint;
  unicodeInput : const unicode;
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	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 described above, byte strings can be as much as six times longer than the number of characters in the input string (or three times longer than the number of bytes in 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 esuniub.c (../../../samplecode/clib_sample/intl_esuniub/esuniub.c.html) for sample code.

See Also

NWUSLenByteToUnicode (page 101), NWUSUnicodeToUntermBytePath (page 118)

NWUSUnicodeToUntermBytePath

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToUntermBytePath  (
   pnuint8                byteOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSUnicodeToUntermBytePath
  (byteOutput : pnuint8;
  outputBufferLen : nuint;
  unicodeInput : const unicode;
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	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 described above, byte strings can be as much as six times longer than the number of characters in the input string (or three times longer than the number of bytes in 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 esuniubp.c (../../../samplecode/clib_sample/intl_esuniubp/esuniubp.c.html) for sample code.

See Also

NWUSLenByteToUnicodePath (page 103), NWUSUnicodeToUntermByte (page 116)

NWUSUnicodeToUpperCase

Converts a NULL-terminated Unicode string to Unicode upper case characters

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUSUnicodeToUpperCase  (
   punicode               upperCaseOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUSUnicodeToUpperCase
  (upperCaseOutput : punicode;
  outputBufferLen : nuint;
  unicodeInput : const unicode;
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDEE	NWU_BUFFER_FULL
0xFDE0	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 esuniup.c (../../../samplecode/clib_sample/intl_esuniup/esuniup.c.html) for sample code.

NWUXByteToUnicode

Converts a NULL-terminated byte string into a Unicode string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXByteToUnicode  (
   pCONVERT              byteUniHandle,
   punicode              unicodeOutput,
   nuint                 outputBufferLen,
   const nuint8 N_FAR   *byteInput,
   pnuint                actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXByteToUnicode
  (byteUniHandle : pCONVERT;
  unicodeOutput : punicode;
  outputBufferLen : nuint
  byteInput : const nuint8;
  actualLength : pnuint
): nint;
```

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 bytes (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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEC	NWU_UNMAPPABLE_CHAR
0xFDEE	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 exbuni.c (../../../samplecode/clib_sample/intl_exbuni/exbuni.c.html) for sample code.

See Also

NWUXByteToUnicodePath (page 124)

NWUXByteToUnicodePath

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXByteToUnicode  (
   pCONVERT              byteUniHandle,
   punicode              unicodeOutput,
   nuint                 outputBufferLen,
   const nuint8 N_FAR   *byteInput,
   pnuint                actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXByteToUnicode
  (byteUniHandle : pCONVERT;
  unicodeOutput : punicode;
  outputBufferLen : nuint
  byteInput : const nuint8;
  actualLength : pnuint
): nint;
```

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 bytes (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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEC	NWU_UNMAPPABLE_CHAR
0xFDEE	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 exbunip.c (../../../samplecode/clib_sample/intl_exbunip/exbunip.c.html) for sample code.

See Also

NWUXByteToUnicode (page 122)

NWUXEnableOemEuro

Enables a Unicode converter to map the Unicode Euro Currency Symbol character to OEM codepages.

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
N_GLOBAL_LIBRARY(nint) NWUXEnableOemEuro (
   pCONVERT   convert);
```

Pascal Syntax

```
uses netwin32

Function NWUXEnableOemEuro
  (convert : pCONVERT
): nint;
```

Parameters

convert

(IN) Points to the Unicode converter handle.

Return Values

0	N_SUCCESS
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDED	NWU_RANGE_ERROR

Remarks

The NetWare client converts file and directory names from ANSI to OEM codepages before sending them to the server. The client performs this conversion only for Windows applications, Windows Explorer, NWAdmin, etc.; there is no conversion for DOS applications.

Since the Euro character is not supported in any of the OEM codepages and it is supported in the ANSI codepages, Novell chose 0xCC to represent the character on the server. The line-drawing character that previously occupied that space is essentially removed from the character set, since line-drawing characters cannot be used in file names. (The line-drawing characters can be used to create files in DOS, but not in Windows.)

See Also

See oemeuro.c (../../../samplecode/clib_sample/intl_oemeuro/oemeuro.c.html) for sample code.

NWUXGetByteFunctions

Returns the functions used for handling unmappable bytes

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetByteFunctions  (
   pCONVERT   byteUniHandle,
   NMBYTE    *noMapByteFunc,
   SCBYTE    *scanByteFunc,
   PRBYTE    *parseByteFunc);
```

Pascal Syntax

```
uses netwin32

Function NWUXGetByteFunctions
  (byteUniHandle : pCONVERT;
  noMapByteFunc : NMBYTE;
  scanByteFunc : SCBYTE
  parseByteFunc : PRBYTE
): nint;
```

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 exgbfunc.c (../../../samplecode/clib_sample/intl_exgbfunc/exgbfunc.c.html) for sample code.

See Also

NWUXGetNoMapAction (page 132), NWUXGetUniFunctions (page 140), NWUXGetSubByte (page 136), NWUXGetSubUni (page 138), NWUXSetNoMapAction (page 154), NWUXSetByteFunctions (page 152), NWUXSetUniFunctions (page 162), NWUXSetSubByte (page 158), NWUXSetSubUni (page 160)

NWUXGetCharSize

Returns the character size (1 or 2) of the next character in the specified byte string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetCharSize  (
   pCONVERT               byteUniHandle,
   const unicode N_FAR   *byteInput,
   pnuint                 charSize);
```

Pascal Syntax

```
uses netwin32

Function NWUXGetCharSize
  (byteUniHandle : pCONVERT;
  byteInput : const unicode;
  charSize : pnuint
): nint;
```

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

0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE

0xFDEB	NWU_HANDLE_MISMATCH

Remarks

NWUXGetCharSize is provided so that user-written Scan and Parse functions are able to examine a byte string.

NOTE: NWCharType (Internationalization) interprets the specified character according to the native system codepage. NWUXGetCharSize uses the codepage of the specified converter.

See excharsz.c (../../../samplecode/clib_sample/intl_excharsz/excharsz.c.html) for sample code.

NWUXGetNoMapAction

Returns the actions to follow when an unmappable byte sequence and an unmappable Unicode character are found

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetNoMapAction  (
   pCONVERT   byteUniHandle,
   pnint      noMapByteAction,
   pnint      noMapUniAction);
```

Pascal Syntax

```
uses netwin32

Function NWUXGetNoMapAction
  (byteUniHandle : pCONVERT;
  noMapByteAction : pnint;
  noMapUniAction : pnint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	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 exgnomap.c (../../../samplecode/clib_sample/intl_exgnomap/exgnomap.c.html) for sample code.

See Also

NWUXGetByteFunctions (page 128), NWUXGetUniFunctions (page 140), NWUXGetSubByte (page 136), NWUXGetSubUni (page 138), NWUXSetNoMapAction (page 154), NWUXSetByteFunctions (page 152), NWUXSetUniFunctions (page 162), NWUXSetSubByte (page 158), NWUXSetSubUni (page 160)

NWUXGetScanAction

Returns the status of current scan/parse functions

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetScanAction  (
   pCONVERT   byteUniHandle,
   pnint      scanByteAction,
   pnint      scanUniAction);
```

Pascal Syntax

```
uses netwin32

Function NWUXGetScanAction
  (byteUniHandle : pCONVERT;
  scanByteByteAction : pnint;
  scanUniAction : pnint
): nint;
```

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

0x0000	SUCCESSFUL

0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDED	NWU_RANGE_ERROR

Remarks

The `scanByteAction` and `scanUniAction` parameters return one of the following values:

NWU_ENABLED	Scan/parse functions are enabled
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 exgscan.c (../../../samplecode/clib_sample/intl_exgscan/exgscan.c.html) for sample code.

See Also

NWUXSetScanAction (page 156), NWUXGetByteFunctions (page 128), NWUXGetUniFunctions (page 140)

NWUXGetSubByte

Returns the substitution byte for the specified converter

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetSubByte  (
   pCONVERT   byteUniHandle,
   pnuint8    substituteByte);
```

Pascal Syntax

```
uses netwin32

Function NWUXGetSubByte
  (byteUniHandle : pCONVERT;
  substituteByte : pnuint8
): nint;
```

Parameters

byteUniHandle

(IN) Points to the handle of the converter to be used.

substituteByte

(OUT) Points to the current substitution byte (see “Substitution Characters” on page 24).

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH

Remarks

See exgsubb.c (../../../samplecode/clib_sample/intl_exgsubb/exgsubb.c.html) for sample code.

See Also

NWUXGetNoMapAction (page 132), NWUXGetByteFunctions (page 128), NWUXGetSubUni (page 138), NWUXGetUniFunctions (page 140), NWUXSetNoMapAction (page 154), NWUXSetByteFunctions (page 152), NWUXSetUniFunctions (page 162), NWUXSetSubByte (page 158), NWUXSetSubUni (page 160)

NWUXGetSubUni

Returns the current substitution Unicode character for the converter

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUGetSubUni   (
   pCONVERT   byteUniHandle,
   punicode   substituteUni);
```

Pascal Syntax

```
uses netwin32

Function NWUGetSubUni
  (byteUniHandle : pCONVERT;
  substituteUni : punicode
): nint;
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

substituteUni

(OUT) Points to the current substitution character (see “Substitution Characters” on page 24).

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH

Remarks

See exgsubu.c (../../../samplecode/clib_sample/intl_exgsubu/exgsubu.c.html) for sample code.

See Also

NWUXGetByteFunctions (page 128), NWUXGetNoMapAction (page 132), NWUXGetSubByte (page 136), NWUXGetUniFunctions (page 140), NWUXSetByteFunctions (page 152), NWUXSetNoMapAction (page 154), NWUXSetSubByte (page 158), NWUXSetSubUni (page 160), NWUXSetUniFunctions (page 162)

NWUXGetUniFunctions

Returns the functions used for handling unmappable Unicode characters

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXGetUniFunctions  (
   pCONVERT   byteUniHandle,
   NMUNI     *noMapUniFunc,
   SCUNI     *scanUniFunc,
   PRUNI     *parseUniFunc);
```

Pascal Syntax

```
uses netwin32

Function NWUXGetUniFunctions
  (byteUniHandle : pCONVERT;
  noMapUniFunc : NMUNI;
  scanUniFunc : SCUNI;
  parseUniFunc : PRUNI
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	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 exgufunc.c (../../../samplecode/clib_sample/intl_exgufunc/exgufunc.c.html) for sample code.

See Also

NWUXGetByteFunctions (page 128), NWUXGetNoMapAction (page 132), NWUXGetSubByte (page 136), NWUXGetSubUni (page 138), NWUXSetByteFunctions (page 152), NWUXSetNoMapAction (page 154), NWUXSetSubByte (page 158), NWUXSetSubUni (page 160), NWUXSetUniFunctions (page 162)

NWUXLenByteToUnicode

Converts a length-specified byte string into a Unicode string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLenByteToUnicode (
   pCONVERT              byteUniHandle,
   punicode              unicodeOutput,
   nuint                 outputBufferLen,
   const nuint8 N_FAR   *byteInput,
   nuint                 inLength,
   pnuint                actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXLenByteToUnicode
  (byteUniHandle : pCONVERT;
  unicodeOutput : punicode;
  byteInput : const nuint8;
  inLength : nuint;
  actualLength : pnuint
): nint;
```

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 bytes (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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEC	NWU_UNMAPPABLE_CHAR
0xFDEE	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, see “Length-Specified Byte String Conversion” on page 25.

See exlbuni.c (../../../samplecode/clib_sample/intl_exlbuni/exlbuni.c.html) for sample code.

See Also

NWUXLenByteToUnicodePath (page 144), NWUXUnicodeToUntermByte (page 170)

NWUXLenByteToUnicodePath

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLenByteToUnicode  (
   pCONVERT              byteUniHandle,
   punicode              unicodeOutput,
   nuint                 outputBufferLen,
   const nuint8 N_FAR   *byteInput,
   nuint                 inLength,
   pnuint                actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXLenByteToUnicode
  (byteUniHandle : pCONVERT;
  unicodeOutput : punicode;
  outputBufferLen : nuint;
  byteInput : const nuint8;
  inLength : nuint;
  actualLength : pnuint
): nint;
```

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 bytes (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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEC	NWU_UNMAPPABLE_CHAR
0xFDEE	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” on page 25.

See exlbunip.c (../../../samplecode/clib_sample/intl_exlbunip/exlbunip.c.html) for sample code.

See Also

NWUXLenByteToUnicode (page 142), NWUXUnicodeToUntermBytePath (page 172)

NWUXLoadByteUnicodeConverter

Locates and loads a converter to convert between Unicode and the specified code page

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLoadByteUnicodeConverter  (
   nuint             codepage,
   pCONVERT N_FAR   *byteUniHandle);
```

Pascal Syntax

```
uses netwin32

Function NWUXLoadByteUnicodeConverter
  (codepage : nuint;
  byteUniHandle : pCONVERT
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE1	NWU_CONVERTER_NOT_FOUND
0xFDE2	NWU_TOO_MANY_FILES (NIOS only)
0xFDE3	NWU_NO_PERMISSION (NIOS only)
0xFDE4	NWU_OPEN_FAILED (NIOS only)

0xFDE5	NWU_READ_FAILED (NIOS only)
0xFDE6	NWU_OUT_OF_MEMORY
0xFDE7	NWU_CANT_LOAD_CONVERTER
0xFDE8	NWU_CONVERTER_CORRUPT

For an explanation of the NWU_CONVERTER_NOT_FOUND error, see "NWU_CONVERTER_NOT_FOUND Error" on page 23

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 other functions.

See exldbu.c (../../../samplecode/clib_sample/intl_exldbu/exldbu.c.html) for sample code.

See Also

NWUXLoadCaseConverter (page 149), NWUXUnloadConverter (page 174)

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: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXLoadCaseConverter  (
   nuint            caseFlag,
   pCONVERT N_FAR  *caseHandle);
```

Pascal Syntax

```
uses netwin32

Function NWUXLoadCaseConverter
  (caseFlag : nuint;
  caseHandle : pCONVERT
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE1	NWU_CONVERTER_NOT_FOUND

0xFDE2	NWU_TOO_MANY_FILES (NIOS only)
0xFDE3	NWU_NO_PERMISSION (NIOS only)
0xFDE4	NWU_OPEN_FAILED (NIOS only)
0xFDE5	NWU_READ_FAILED (NIOS only)
0xFDE6	NWU_OUT_OF_MEMORY
0xFDE7	NWU_CANT_LOAD_CONVERTER
0xFDE8	NWU_CONVERTER_CORRUPT
0xFDED	NWU_RANGE_ERROR

For an explanation of the NWU_CONVERTER_NOT_FOUND error, see "NWU_CONVERTER_NOT_FOUND Error" on page 23

See exldcase.c (../../../samplecode/clib_sample/intl_exldcase/exldcase.c.html) for sample code.

See Also

NWUXLoadByteUnicodeConverter (page 147), NWUXUnloadConverter (page 174)

NWUXResetConverter

Resets the converter to a default state

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXResetConverter  (
   pCONVERT   convert);
```

Pascal Syntax

```
uses netwin32

Function NWUXResetConverter
  convert : pCONVERT
): nint;
```

Parameters

convert

(IN) Points to the handle of the converter to be reset.

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH

Remarks

See exreset.c (../../../samplecode/clib_sample/intl_exreset/exreset.c.html) for sample code.

NWUXSetByteFunctions

Specifies the functions to be used to handle unmappable bytes and special byte sequences during byte-to-unicode conversion

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetByteFunctions  (
   pCONVERT   byteUniHandle,
   NMBYTE     noMapByteFunc,
   SCBYTE     scanByteFunc,
   PRBYTE     parseByteFunc);
```

Pascal Syntax

```
uses netwin32

Function NWUXSetByteFunctions
  (byteUniHandle : pCONVERT;
  noMapByteFunc : NMBYTE;
  scanByteFunc : SCBYTE;
  parseByteFunc : PRBYTE
): nint;
```

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 exsbfunc.c (../../../samplecode/clib_sample/intl_exsbfunc/exsbfunc.c.html) for sample code.

See Also

NWUXGetByteFunctions (page 128), NWUXGetUniFunctions (page 140), NWUXGetSubByte (page 136), NWUXGetSubUni (page 138), NWUXResetConverter (page 151), NWUXSetNoMapAction (page 154), NWUXSetUniFunctions (page 162), NWUXSetSubByte (page 158), NWUXSetSubUni (page 160)

NWUXSetNoMapAction

Sets the actions to follow when an unmappable byte or an unmappable Unicode character is found

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetNoMapAction  (
   pCONVERT   byteUniHandle,
   nint       noMapByteAction,
   nint       noMapUniAction);
```

Pascal Syntax

```
uses netwin32

Function NWUXSetNoMapAction
  (byteUniHandle : pCONVERT;
  noMapByteFunc : nint;
  noMapUniFunc : nint;
): nint;
```

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_SUBSTITUTE	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_UNCHANGED_ACTION	Leaves the current setting unchanged.

See exsnomap.c (../../../samplecode/clib_sample/intl_exsnomap/exsnomap.c.html) for sample code.

See Also

NWUXGetNoMapAction (page 132), NWUXGetByteFunctions (page 128), NWUXGetUniFunctions (page 140), NWUXGetSubByte (page 136), NWUXGetSubUni (page 138), NWUXSetByteFunctions (page 152), NWUXSetUniFunctions (page 162), NWUXSetSubByte (page 158), NWUXSetSubUni (page 160)

NWUXSetScanAction

Enables or disables the current scan/parse functions

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetScanAction  (
   pCONVERT   byteUniHandle,
   nint       scanByteAction,
   nint       scanUniAction);
```

Pascal Syntax

```
uses netwin32

Function NWUXSetScanAction
  (byteUniHandle : pCONVERT;
  scanByteFunc : nint;
  ScanUniFunc : nint;
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE

0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDED	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 exsscan.c (../../../samplecode/clib_sample/intl_exsscan/exsscan.c.html) for sample code.

See Also

NWUXGetScanAction (page 134), NWUXGetByteFunctions (page 128), NWUXGetUniFunctions (page 140)

NWUXSetSubByte

Specifies the substitution byte for the converter

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetSubByte  (
   pCONVERT   byteUniHandle,
   nuint8     substituteByte);
```

Pascal Syntax

```
uses netwin32

Function NWUXSetSubByte
  (byteUniHandle : pCONVERT;
  substituteByte : nuint8
): nint;
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

substituteByte

(IN) Specifies the new substitution byte.

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	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 exssubb.c (../../../samplecode/clib_sample/intl_exssubb/exssubb.c.html) for sample code.

See Also

NWUXGetByteFunctions (page 128), NWUXGetNoMapAction (page 132), NWUXGetSubByte (page 136), NWUXGetSubUni (page 138), NWUXGetUniFunctions (page 140), NWUXSetByteFunctions (page 152), NWUXSetNoMapAction (page 154), NWUXSetUniFunctions (page 162), NWUXSetSubUni (page 160)

NWUXSetSubUni

Specifies the substitution character for the converter

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetSubUni  (
   pCONVERT   byteUniHandle,
   Unicode    subsituteUni);
```

Pascal Syntax

```
uses netwin32

Function NWUXSetSubUni
  (byteUniHandle : pCONVERT;
  substituteUni : Unicode
): nint;
```

Parameters

byteUniHandle

(IN) Points to the converter handle.

substituteUni

(OUT) Receives the new substitution character.

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	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 exssubu.c (../../../samplecode/clib_sample/intl_exssubu/exssubu.c.html) for sample code.

See Also

NWUXGetByteFunctions (page 128), NWUXGetNoMapAction (page 132), NWUXGetSubByte (page 136), NWUXGetSubUni (page 138), NWUXGetSubUni (page 138), NWUXGetUniFunctions (page 140), NWUXSetByteFunctions (page 152), NWUXSetNoMapAction (page 154), NWUXSetUniFunctions (page 162), NWUXSetSubByte (page 158)

NWUXSetUniFunctions

Specifies the functions to be used to handle unmappable Unicode characters and special Unicode sequences during unicode-to-byte conversion

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXSetUniFunctions  (
   pCONVERT   byteUniHandle,
   NMUNI      noMapUniFunc,
   SCUNI      scanUniFunc,
   PRUNI      parseUniFunc);
```

Pascal Syntax

```
uses netwin32

Function NWUXSetUniFunctions
  (byteUniHandle : pCONVERT;
  noMapUniFunc : NMUNI;
  scanUniFunc : SCUNI;
  parseUniFunc : PRUNI
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	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 exsufunc.c (../../../samplecode/clib_sample/intl_exsufunc/exsufunc.c.html) for sample code.

See Also

NWUXGetByteFunctions (page 128), NWUXGetUniFunctions (page 140), NWUXGetSubByte (page 136), NWUXGetSubUni (page 138), NWUXResetConverter (page 151), NWUXSetNoMapAction (page 154), NWUXSetByteFunctions (page 152), NWUXSetSubByte (page 158), NWUXSetSubUni (page 160)

NWUXUnicodeToByte

Converts a Unicode string into a NULL-terminated byte string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToByte  (
   pCONVERT               byteUniHandle,
   pnuint8                byteOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXUnicodeToByte
  (byteUniHandle : pCONVERT;
  byteOutput : pnuint8;
  outputBufferLen : nuint
  unicodeInput : const unicode
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEC	NWU_UNMAPPABLE_CHAR
0xFDEE	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 exunib.c (../../../samplecode/clib_sample/intl_exunib/exunib.c.html) for sample code.

See Also

NWUXUnicodeToBytePath (page 166), NWUXUnicodeToCase (page 168)

NWUXUnicodeToBytePath

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

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToBytePath  (
   pCONVERT               byteUniHandle,
   pnuint8                byteOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXUnicodeToBytePath
  (byteUniHandle : pCONVERT;
  byteOutput : pnuint8;
  outputBufferLen : nuint
  unicodeInput : const unicode
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEC	NWU_UNMAPPABLE_CHAR
0xFDEE	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 exunibp.c (../../../samplecode/clib_sample/intl_exunibp/exunibp.c.html) for sample code.

See Also

NWUXSetNoMapAction (page 154), NWUXUnicodeToByte (page 164), NWUXUnicodeToCase (page 168)

NWUXUnicodeToCase

Converts a NULL-terminated Unicode string to upper case, lower case, or title case, depending on the converter pointed to

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToCase  (
   pCONVERT               caseHandle,
   punicode               monocasedOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXUnicodeToCase
  (caseHandle : pCONVERT;
  monocasedOutput : punicode;
  outputBufferLen : nuint
  unicodeInput : const unicode
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEE	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 exucase.c (../../../samplecode/clib_sample/intl_exucase/exucase.c.html) for sample code.

See Also

NWUXLoadCaseConverter (page 149)

NWUXUnicodeToUntermByte

Converts a Unicode string into an unterminated byte string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToUntermByte  (
   pCONVERT               byteUniHandle,
   pnuint8                byteOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXUnicodeToUntermByte
  (byteUniHandle : pCONVERT;
  byteOutput : pnuint8;
  outputBufferLen : nuint
  unicodeInput : const unicode
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEC	NWU_UNMAPPABLE_CHAR
0xFDEE	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 exuniub.c (../../../samplecode/clib_sample/intl_exuniub/exuniub.c.html) for sample code.

See Also

NWUXLenByteToUnicode (page 142), NWUXUnicodeToUntermBytePath (page 172)

NWUXUnicodeToUntermBytePath

Converts a Unicode file path string into an unterminated byte string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnicodeToUntermBytePath  (
   pCONVERT               byteUniHandle,
   pnuint8                byteOutput,
   nuint                  outputBufferLen,
   const unicode N_FAR   *unicodeInput,
   pnuint                 actualLength);
```

Pascal Syntax

```
uses netwin32

Function NWUXUnicodeToUntermBytePath
  (byteUniHandle : pCONVERT;
  byteOutput : pnuint8;
  outputBufferLen : nuint
  unicodeInput : const unicode
  actualLength : pnuint
): nint;
```

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

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	NWU_BAD_HANDLE
0xFDEB	NWU_HANDLE_MISMATCH
0xFDEC	NWU_UNMAPPABLE_CHAR
0xFDEE	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 exuniubp.c (../../../samplecode/clib_sample/intl_exuniubp/exuniubp.c.html) for sample code.

See Also

NWUXLenByteToUnicodePath (page 144), NWUXUnicodeToUntermByte (page 170)

NWUXUnloadConverter

Unloads a converter and releases all associated resources

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <nunicode.h>

N_EXTERN_LIBRARY(nint) NWUXUnloadConverter  (
   pCONVERT   converterHandle);
```

Pascal Syntax

```
uses netwin32

Function NWUXUnloadConverter
  (convertHandle : pCONVERT
): nint;
```

Parameters

converterHandle

(IN) Points to the converter handle to release.

Return Values

0x0000	SUCCESSFUL
0xFDE9	NWU_NULL_HANDLE
0xFDEA	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 exunl.c (../../../samplecode/clib_sample/intl_exunl/exunl.c.html) for sample code.

See Also

NWUXLoadByteUnicodeConverter (page 147), NWUXLoadCaseConverter (page 149)

unicat

Appends a copy of a specified string to the end of another string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unicat  (
   punicode               s1,
   const unicode N_FAR   *s2);
```

Pascal Syntax

```
uses netwin32

Function unicat
  (s1 : punicode;
   s2 : const unicode
) : 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)
```

For sample code, see Section 3.11, “Example: unicat,” on page 51.

See Also

unincat (page 189)

unichr

Finds the first occurrence of a given character in a specified string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unichr  (
   const unicode N_FAR  *s,
   unicode               c);
```

Pascal Syntax

```
uses netwin32

Function unichr
  (s : const unicode;
   c : unicode
) : punicode;
```

Parameters

s

(IN) Points to the string to be scanned.

c

(IN) Specifies the character to be found.

Return Values

Non-NULL	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.

For sample code, see Section 3.12, “Example: unichr,” on page 51.

See Also

unirchr (page 203), unistr (page 211)

unicmp

Compares two Unicode strings for differences

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) unicmp  (
   const unicode N_FAR  *s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function unicmp
  (s1 : const unicode;
   s2 : const unicode
) : 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.

For sample code, see Section 3.13, “Example: unicmp,” on page 52.

See Also

uniicmp (page 186), unincmp (page 191)

unicpy

Copies from one specified string to another

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unicpy  (
   punicode              s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function unicpy
  (s1 : punicode;
   s2 : const unicode
) : 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.

For sample code, see Section 3.14, “Example: unicpy,” on page 52.

See Also

unincpy (page 193), unipcpy (page 201)

unicspn

Scans a specified string for the initial segment not containing any subset of the given set of characters

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(size_t) unicspn  (
   const unicode N_FAR  *s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function unicspn
  (s1 : const unicode;
   s2 : const unicode
) : 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.

For sample code, see Section 3.15, “Example: unicspn,” on page 52.

See Also

unispn (page 209), unipbrk (page 199)

uniicmp

Compares the lower case versions of two different strings (using the character values, not the collation value) for differences

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) uniicmp  (
   const unicode N_FAR  *s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function uniicmp
  (s1 : const unicode;
   s2 : const unicode
) : 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.

For sample code, see Section 3.16, “Example: uniicmp,” on page 53.

unilen

Returns the number of unicode characters in a string (not counting the NULL terminator)

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(size_t) unilen  (
   const unicode N_FAR  *s);
```

Pascal Syntax

```
uses netwin32

Function unilen
  (s : const unicode
) : 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.

For sample code, see Section 3.17, “Example: unilen,” on page 53.

See Also

unisize (page 208)

unincat

Copies the specified number of characters from one string to the end of another string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unincat  (
   punicode              s1,
   const unicode N_FAR  *s2,
   size_t                n);
```

Pascal Syntax

```
uses netwin32

Function unincat
  (s1 : punicode;
   s2 : const unicode;
   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.

For sample code, see Section 3.18, “Example: unincat,” on page 53.

See Also

unicat (page 176)

unincmp

Compares a specified number of characters of two Unicode strings

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) unincmp  (
   const unicode N_FAR  *s1,
   const unicode N_FAR  *s2,
   size_t                len);
```

Pascal Syntax

```
uses netwin32

Function unincmp
  (s1  : const unicode;
   s2  : const unicode;
   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.

For sample code, see Section 3.19, “Example: unincmp,” on page 54.

See Also

unicmp (page 180), uninicmp (page 195)

unincpy

Copies a specified number of characters from one string to another

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unincpy  (
   punicode              s1,
   const unicode N_FAR  *s2,
   size_t                n);
```

Pascal Syntax

```
uses netwin32

Function unincpy
  (s1 : punicode;
   s2 : const unicode;
   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.

For sample code, see Section 3.22, “Example: unipcpy,” on page 55.

See Also

unicpy (page 182), unipcpy (page 201)

uninicmp

Compares a specified number of characters (using the character values, not the collation value) of the lower case versions of two strings

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(nint) uninicmp  (
   const unicode N_FAR  *s1,
   const unicode N_FAR  *s2,
   size_t                len);
```

Pascal Syntax

```
uses netwin32

Function uninicmp
  (s1  : const unicode;
   s2  : const unicode;
   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.

For sample code, see Section 3.20, “Example: uninicmp,” on page 54.

See Also

uniicmp (page 186)

uninset

Copies a given character into the first specified number of characters of a string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) uninset  (
   punicode   s,
   unicode    c,
   size_t     n);
```

Pascal Syntax

```
uses netwin32

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.

For sample code, see Section 3.21, “Example: uninset,” on page 55.

See Also

uniset (page 206)

unipbrk

Scans a specified string for a given set of characters

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unipbrk  (
   const unicode N_FAR  *s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function unipbrk
  (s1 : const unicode;
   s2 : const unicode
) : 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.

For sample code, see Section 3.23, “Example: unipbrk,” on page 56.

See Also

unicspn (page 184), unispn (page 209)

unipcpy

Copies one string to another up to the null-termination

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unipcpy  (
   punicode              s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function unipcpy
  (s1 : punicode;
   s2 : const unicode
) : 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.

For sample code, see Section 3.22, “Example: unipcpy,” on page 55.

See Also

unicpy (page 182), unincpy (page 193)

unirchr

Scans a string in the reverse direction, searching for the last occurrence of a given character

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unirchr  (
   const unicode N_FAR  *s,
   unicode               c);
```

Pascal Syntax

```
uses netwin32

Function unirchr
  (s : const unicode;
   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 occurrence 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.

For sample code, see Section 3.24, “Example: unirchr,” on page 56.

See Also

unichr (page 178)

unirev

Reverses the order of all characters in a string, except for the null terminating character

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unirev  (
   punicode   s);
```

Pascal Syntax

```
uses netwin32

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.

For sample code, see Section 3.25, “Example: unirev,” on page 56.

uniset

Sets all characters in a string to a specified character

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) uniset  (
   punicode   s,
   unicode    c);
```

Pascal Syntax

```
uses netwin32

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.

For sample code, see Section 3.26, “Example: uniset,” on page 57.

See Also

uninset (page 197)

unisize

Determines the size in bytes (not characters) the current string occupies including the 2-byte NULL terminator

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(size_t) unisize  (
   const unicode N_FAR  *s);
```

Pascal Syntax

```
uses netwin32

Function unisize
  (s : const unicode
) : 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.

Remarks

For sample code, see Section 3.27, “Example: unisize,” on page 57.

See Also

unilen (page 188)

unispn

Finds the first segment of a string consisting entirely of characters from another string

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(size_t) unispn  (
   const unicode N_FAR  *s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function unispn
  (s1 : const unicode;
   s2 : const unicode
) : 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.

For sample code, see Section 3.28, “Example: unispn,” on page 58.

See Also

unicspn (page 184), unipbrk (page 199)

unistr

Scans a string for the first occurrence of a specified substring

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unistr  (
   const unicode N_FAR  *s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function unistr
  (s1 : const unicode;
   s2 : const unicode
) : punicode;
```

Parameters

s1

(IN) Points to the string to be scanned.

s2

(IN) Points to the string to be located.

Return Values

Non-NULL	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.

For sample code, see Section 3.29, “Example: unistr,” on page 58.

See Also

unichr (page 178)

unitok

Searches a string for tokens separated by specified delimiters

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows95, Windows 98

Library: Cross-Platform Localization (LOC*.*)

Service: Unicode

Syntax

```
#include <unicode.h>

N_EXTERN_LIBRARY(punicode) unitok  (
   punicode              s1,
   const unicode N_FAR  *s2);
```

Pascal Syntax

```
uses netwin32

Function unitok
  (s1 : punicode;
   s2 : const unicode
) : punicode;
```

Parameters

s1

(IN) Points to the string to parse.

s2

(IN) Points to the delimiter values.

Return Values

If a delimiter is found, returns a pointer to the first byte of a token. On subsequent iterations if no delimiter is found, returns a null pointer.

If the string does not contain any of the delimiters specified in `s2`, returns a pointer to the string. All of the string is considered to be a token.

Remarks

The unitok function 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 string parameter and writes a null character into the string parameter immediately following the returned token. The unitok function sets the string parameter to a null pointer which allows unitok, on subsequent calls, to search through the string and return successive tokens, until no tokens remain. Because unitok can modify the original string, that string should be duplicated if the string is to be reused.

The unitok function corresponds to the strtok function.

For sample code, see Section 3.30, “Example: unitok,” on page 58.

Revision History

A

The following table outlines all the changes that have been made to the Unicode documentation (in reverse chronological order):

October 5, 2005	Transitioned to revised Novell documentation standards.
March 2, 2005	Fixed the preface and legal information.
July 30, 2003	Updated the documentation for unitok (page 213).
March 2002	Updated Pascal syntax for NWUSUnicodeToByte (page 110).
February 2002	Updated links.
September 2001	Added support for NetWare 6.x to documentation.
June 2001	Changed the descriptions of `outputBufferLen` in NWUXByteToUnicode (page 122), NWUXByteToUnicodePath (page 124), NWUXLenByteToUnicode (page 142), and NWUXLenByteToUnicodePath (page 144).
	Updated tables.
February 2001	Added Delphi (Pascal) syntax to functions where missing. Corrected return error code (-500 changed to -506) for the NWLUTF8ToUnicodeSize (page 83) function.
July 2000	Removed documentation for NWUXLoadCollationConverter, NWUXLoadNormalizeConverter, NWUXUnicodeToCollation, and NWUXUnicodeToNormalized because these functions were never implemented.
May 2000	Added documentation for NWUXEnableOemEuro (page 126).
	Added sample code links in NWLUnicodeToUTF8 (page 78), NWLUnicodeToUTF8Size (page 80), NWLUTF8ToUnicode (page 81), and NWLUTF8ToUnicodeSize (page 83).
March 2000	Added NWLUnicodeToUTF8 (page 78), NWLUnicodeToUTF8Size (page 80), NWLUTF8ToUnicode (page 81), and NWLUTF8ToUnicodeSize (page 83).
	Changed nwnunicode.h to nunicode.h and nwunicode.h to unicode.h in the "About This Guide" on page 11. Also, removed asterisk (pointer indicator) from first parameter in syntax of unincpy (page 193).
	Removed NetWare 3.x references from all "Functions" on page 61. Also, fixed wording about the length of the input string due to default conversion behavior found in several Unicode functions.
January 2000	Added const to syntax of input pointers.
November 1999	Added library information to each Unicode Function.