

Novell Developer Kit

www.novell.com

October 5, 2005

PROGRAM MANAGEMENT

N

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.

Exceleator 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	13
1 Bit Array Functions	15
BitClear	16
BitSet	17
BitTest	18
BitTestAndClear	19
BitTestAndSet	20
ScanBits	21
ScanClearedBits	23
2 Character Manipulation Functions	25
isalnum	26
isalpha	27
isascii	29
iscntrl	31
isdigit	33
isgraph	35
islower	36
isprint	37
ispunct	38
isspace	39
isupper	41
isxdigit	42
mblen	43
mbstowcs	44
mbtowc	45
tolower	47
toupper	49
wcstombs	51
wctomb	52
3 Data Manipulation Functions	53
bsearch	54
IntSwap	56
LongSwap	57
_rotl	58
_rorl	59
NWLongSwap	60
NWWordSwap	61
offsetof	62
qsort	63
_rotl	65
_rorl	66

4	Library Concepts	67
4.1	Unique Problems of Library NLMs	67
4.2	Library API Example	67
4.3	Library Function List	71
5	Library Functions	73
	DeregisterLibrary	74
	GetDataAreaPtr	75
	RegisterLibrary	77
	SaveDataAreaPtr	80
6	Mathematical Computation Functions	83
	abs	85
	acos	86
	asin	87
	atan	88
	atan2	89
	Bessel Functions	90
	cabs	92
	ceil	94
	cos	95
	cosh	96
	div	97
	exp	99
	fabs	100
	floor	101
	fmod	103
	frexp	105
	hypot	107
	labs	108
	ldexp	109
	ldiv	111
	log	113
	log10	114
	matherr	115
	max	117
	min	118
	modf	119
	pow	121
	rand, rand_r	123
	RegisterMatherrHandler	124
	sin	126
	sinh	127
	sqrt	128
	srand	129
	tan	130
	tanh	131

7 Mathematical Computation Structures	133
complex	134
div_t	135
exception	136
ldiv_t	137
8 Memory Allocation Functions	139
alloca	140
calloc	142
free	144
malloc	145
_msize	147
NWGetAllocPageOverhead	148
NWGetAvailableMemory	149
NWGetPageSize	150
NWMemorySizeAddressable	151
__qcalloc	152
__qmalloc	153
__qrealloc	154
realloc	155
stackavail	157
9 Memory Manipulation Functions	159
memchr	160
memcmp	162
memcpy	164
memicmp	166
memmove	168
memset	170
10 String Conversion Functions	173
ASCIIZToLenStr	174
ASCIIZToMaxLenStr	175
atof	177
atoi	178
atol	179
ecvt	180
fcvt	182
gcvt	184
itoa	186
LenToASCIIZStr	188
ltoa	189
strtod	191
strtoi	193
strtol	195
strtoul	197
ultoa	199
utoa	201

11 String Manipulation Functions	203
LenStrCat	205
LenStrCmp	206
LenStrCpy	207
sprintf	208
sscanf	210
strcasecmp	212
strcat	213
strchr	215
strcmp	217
strcmpi	219
strcoll	221
strcpy	223
strcspn	224
strdup	226
strerror	227
stricmp	228
strlen	230
strlist	231
strlwr	232
strncasecmp	234
strncat	235
strncmp	237
strncpy	239
strnicmp	241
strnset	243
strpbrk	245
strchr	247
strev	249
strsep	250
strset	251
strspn	253
strstr	255
strtok	257
strupr	259
strxfrm	261
swab	263
swaw	265
vsprintf	267
vsscanf	269
12 Time/Date Manipulation Functions	271
asctime, asctime_r	272
clock	274
_ConvertDOSTimeToCalendar	275
_ConvertTimeToDOS	277
ctime, ctime_r	279
difftime	281
GetClockStatus	283
GetCurrentTicks	285

gmtime, gmtime_r	286
localtime, localtime_r	288
ltime	290
mktime	291
NWPackDate	293
NWPackDateTime	295
NWPackTime	297
NWUnpackDate	298
NWUnpackDateTime	300
NWUnpackTime	302
SecondsToTicks	303
strftime	304
TicksToSeconds	307
time	308
tzset	309

13 Time/Date Manipulation Structures 311

NW_DATE	312
NW_TIME	313
tm	314

14 Variable Length Argument List Functions 317

va_arg	318
va_end	321
va_start	322

A Revision History 323

About This Guide

This documentation contains information directly related to C programming on the NLM™ and NetWare® platforms. Functions and associated information are provided for the following:

- Chapter 1, “Bit Array Functions,” on page 15
- Chapter 2, “Character Manipulation Functions,” on page 25
- Chapter 3, “Data Manipulation Functions,” on page 53
- Chapter 4, “Library Concepts,” on page 67
- Chapter 5, “Library Functions,” on page 73
- Chapter 6, “Mathematical Computation Functions,” on page 83
- Chapter 7, “Mathematical Computation Structures,” on page 133
- Chapter 8, “Memory Allocation Functions,” on page 139
- Chapter 9, “Memory Manipulation Functions,” on page 159
- Chapter 10, “String Conversion Functions,” on page 173
- Chapter 11, “String Manipulation Functions,” on page 203
- Chapter 12, “Time/Date Manipulation Functions,” on page 271
- Chapter 13, “Time/Date Manipulation Structures,” on page 311
- Chapter 14, “Variable Length Argument List Functions,” on page 317

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) (<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)
- Single and Intra-File Services (http://developer.novell.com/ndk/doc/clib/sngl_enu/data/h68b1qom.html)
- Multiple and Inter-File Services (http://developer.novell.com/ndk/doc/clib/mlti_enu/data/hby40vgi.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/cmngxenu/data/hvxfva0i.html>)
- Internationalization (http://developer.novell.com/ndk/doc/clib/intl_enu/data/h70a28iu.html)
- Volume Management (http://developer.novell.com/ndk/doc/clib/vol__enu/data/h6ixme3w.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)
- Server Management (http://developer.novell.com/ndk/doc/clib/srvr_enu/data/hzvjztxz.html)
- Unicode (http://developer.novell.com/ndk/doc/clib/ucod_enu/data/hjg275fp.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 source code projects, visit [Forge](http://forge.novell.com) (<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) (<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.

Bit Array Functions

1

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

- [“BitClear” on page 16](#)
- [“BitSet” on page 17](#)
- [“BitTest” on page 18](#)
- [“BitTestAndClear” on page 19](#)
- [“BitTestAndSet” on page 20](#)
- [“ScanBits” on page 21](#)
- [“ScanClearedBits” on page 23](#)

BitClear

Clears the specified bit

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Bit Array

Syntax

```
#include <nwbitops.h>

void BitClear (
    void    *bitArray,
    LONG    bitNumber);
```

Parameters

bitArray

(IN) Points to the bit array.

bitNumber

(IN) Specifies an index into the bit array.

Return Values

None

Remarks

The `bitArray` parameter specifies the target array. The bit number can be greater than 32, targeting a bit well into the target array.

See Also

[BitSet \(page 17\)](#)

Example

```
#include <nwbitops.h>
void    *bitArray;
LONG    bitNumber;
BitClear (bitArray, bitNumber);
```

BitSet

Sets the target bit

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Bit Array

Syntax

```
#include <nwbitops.h>

void BitSet (
    void    *bitArray,
    LONG    bitNumber);
```

Parameters

bitArray

(IN) Points to the bit array.

bitNumber

(IN) Specifies an index into the bit array.

Return Values

None

Remarks

The `bitArray` parameter specifies the target array. The bit number can be greater than 32, targeting a bit well into the target array.

See Also

[BitClear \(page 16\)](#), [BitTest \(page 18\)](#)

Example

```
#include <nwbitops.h>
void    *bitArray;
LONG    bitNumber;
BitSet (bitArray, bitNumber);
```

BitTest

Determines whether the specified bit is set

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Bit Array

Syntax

```
#include <nwbitops.h>

LONG BitTest (
    void *bitArray,
    LONG bitNumber);
```

Parameters

bitArray

(IN) Points to the bit array.

bitNumber

(IN) Specifies an index into the bit array.

Return Values

This function returns a bit value of 0 if the specified bit is cleared. Otherwise, it returns a value of 1.

Remarks

The `bitArray` parameter specifies the target array. The bit number can be greater than 32, targeting a bit well into the target array.

See Also

[BitClear \(page 16\)](#), [BitSet \(page 17\)](#)

Example

```
#include <nwbitops.h>
LONG bitValue;
void *bitArray;
LONG bitNumber;
bitValue = BitTest (bitArray, bitNumber);
```

BitTestAndClear

Returns the current value of the specified bit and then clears the bit (if the bit was not already cleared)

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Bit Array

Syntax

```
#include <nwbitops.h>

LONG BitTestAndClear (
    void *bitArray,
    LONG bitNumber);
```

Parameters

bitArray

(IN) Points to the bit array.

bitNumber

(IN) Specifies an index into the bit array.

Return Values

This function returns an old bit value of 0 if the specified bit is cleared. Otherwise, it returns a value of 1.

Remarks

The `bitArray` parameter specifies the target array. It can be byte-aligned and can point to an array of up to $2^{32} - 1$ bits.

See Also

[BitTestAndSet \(page 20\)](#)

Example

```
#include <nwbitops.h>
LONG oldBitValue;
void *bitArray;
LONG bitNumber;
oldBitValue = BitTestAndClear (bitArray, bitNumber);
```

BitTestAndSet

Returns the current value of the specified bit and then sets the bit (if the bit was not already set)

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Bit Array

Syntax

```
#include <nwbitops.h>

LONG BitTestAndSet (
    void *bitArray,
    LONG bitNumber);
```

Parameters

bitArray

(IN) Points to the bit array.

bitNumber

(IN) Specifies an index into the bit array.

Return Values

This function returns an old bit value of 0 if the specified bit is cleared. Otherwise, it returns a value of 1.

Remarks

The `bitArray` parameter specifies the target array. It can be byte-aligned and can point to an array of up to $2^{32} - 1$ bits.

See Also

[BitTestAndClear \(page 19\)](#)

Example

```
#include <nwbitops.h>
LONG oldBitValue;
void *bitArray;
LONG bitNumber;
oldBitValue = BitTestAndSet (bitArray, bitNumber);
```

ScanBits

Scans a bit array to find the first bit set

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Bit Array

Syntax

```
#include <nwbitops.h>

LONG ScanBits (
    void    *bitArray,
    LONG    startingBitNumber,
    LONG    totalBitCount);
```

Parameters

bitArray

(IN) Points to the bit array.

startingBitNumber

(IN) Specifies the index number of the bit to start searching on.

totalBitCount

(IN) Specifies the size of the bit array.

Return Values

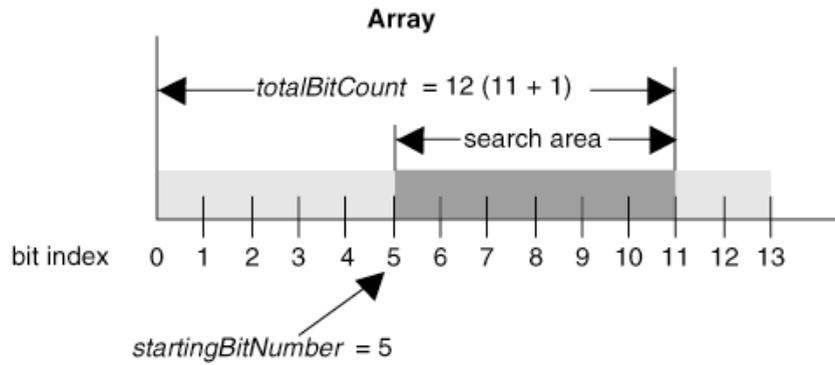
This function returns a bit index (relative to the beginning of the array) to the first bit set, or a value of -1 if no bit is set.

Remarks

The `bitArray` parameter specifies the target array, which can begin on a byte boundary. A bit array can be as large as $2^{32} - 1$ bits in length.

The `totalBitCount` parameter specified the total number of bits from the beginning of the array to the end of the search area. Therefore, if the search area is from bit index 5 to bit index 11, the `startingBitNumber` would be 5 and the `totalBitCount` would be 12 (the bit index +1).

Figure 1-1 Scanning a Bit Array



See Also

[ScanClearedBits \(page 23\)](#)

Example

```
#include <nwbitops.h>
LONG   bitNumber;
void   *bitArray;
LONG   startingBitNumber;
LONG   totalBitCount;
bitNumber = ScanBits (bitArray, startingBitNumber, totalBitCount);
```

ScanClearedBits

Scans a bit array to find the first bit that has been cleared

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Bit Array

Syntax

```
#include <nwbitops.h>

LONG ScanClearedBits (
    void    *bitArray,
    LONG    startingBitNumber,
    LONG    totalBitCount);
```

Parameters

bitArray

(IN) Points to the bit array.

startingBitNumber

(IN) Specifies the index number of the bit to start searching on.

totalBitCount

(IN) Specifies the size of the bit array.

Return Values

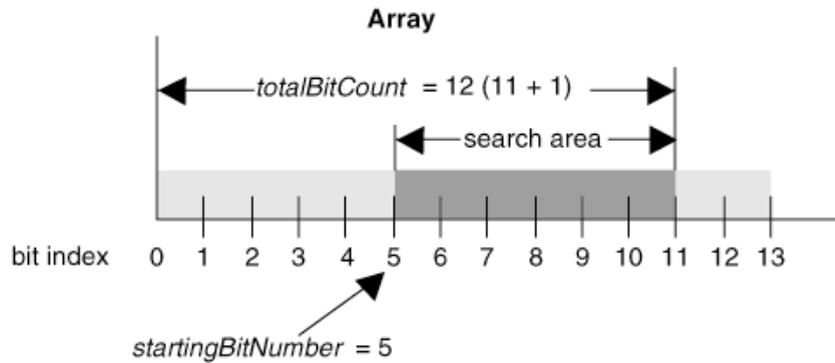
This function returns a bit index (relative to the beginning of the array) to the first bit cleared, or a value of -1 if no bit is cleared.

Remarks

The `bitArray` parameter specifies the target array, which can begin on a byte boundary. A bit array can be as large as $2^{32} - 1$ bits in length.

The `totalBitCount` parameter specified the total number of bits from the beginning of the array to the end of the search area. Therefore, if the search area is from bit index 5 to bit index 11, the `startingBitNumber` would be 5 and the `totalBitCount` would be 12 (the bit index +1).

Figure 1-2 Scanning a Bit Array



See Also

[ScanBits \(page 21\)](#)

Example

```
#include <nwbitops.h>
LONG    bitNumber;
void    *bitArray;
LONG    startingBitNumber;
LONG    totalBitCount;
bitNumber = ScanClearedBits (bitArray, startingBitNumber,
                             totalBitCount);
```

Character Manipulation Functions

2

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

- [“isalnum” on page 26](#)
- [“isalpha” on page 27](#)
- [“isascii” on page 29](#)
- [“isctrl” on page 31](#)
- [“isdigit” on page 33](#)
- [“isgraph” on page 35](#)
- [“islower” on page 36](#)
- [“isprint” on page 37](#)
- [“ispunct” on page 38](#)
- [“isspace” on page 39](#)
- [“isupper” on page 41](#)
- [“isxdigit” on page 42](#)
- [“mblen” on page 43](#)
- [“mbstowcs” on page 44](#)
- [“mbtowc” on page 45](#)
- [“tolower” on page 47](#)
- [“toupper” on page 49](#)
- [“wcstombs” on page 51](#)
- [“wctomb” on page 52](#)

isalnum

Tests for an alphanumeric character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>

int isalnum (
    int  c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

This function or macro returns a value of 0 if the argument is neither an alphabetic character nor a digit. Otherwise, a nonzero value is returned.

Remarks

The `isalnum` function or macro tests if the argument `c` is an alphanumeric character (a to z, A to Z, or 0 to 9). An alphanumeric character is any character for which `isalpha` or `isdigit` is true.

See Also

[isalpha \(page 27\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#)

Example

```
#include <ctype.h>
#include <stdio.h>
main ()
{
    printf ("%d %d \w", isalnum ('Q'), isalnum ('!'));
}
```

produces the following:

```
1 0
```

isalpha

Tests for an alphabetic character (a to z or A to Z) (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>

int isalpha (
    int    c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

isalpha returns a value of 0 if the argument is not an alphabetic character. Otherwise, a nonzero value is returned.

Remarks

The isalpha function or macro tests for an alphabetic character (a to z or A to Z). An alphabetic character is any character for which isupper or islower is true.

See Also

[isalnum \(page 26\)](#), [isctrl \(page 31\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

Example

```
#include <ctype.h>
#include <stdio.h>

main ()
{
    printf ("%d %d %d", isalpha ('2'), isalpha ('Q'), isalpha ('!'));
}
```

produces the following:

0 1 0

isascii

Tests for an ASCII character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Character Manipulation

Syntax

```
#include <ctype.h>

int isascii (
    int c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

The `isascii` function or macro returns a nonzero value when the character is in the range from 0 to 127. Otherwise, a value of 0 is returned.

Remarks

The `isascii` function or macro tests for a character in the range from 0 to 127.

See Also

[isalpha \(page 27\)](#), [isalnum \(page 26\)](#), [isctrl \(page 31\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

Example

```
#include <ctype.h>
#include <stdio.h>

main ()
{
    printf ("%d %d %d \u", isascii ('\u'), isascii ('\f'),
           isascii ('A'));
}
```

produces the following:

1 1 1

iscntrl

Tests for a control character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>

int iscntrl (
    int c);
```

Parameters

c
(IN) Specifies the character to be tested.

Return Values

The `iscntrl` function or macro returns a nonzero value when the argument is a control character. Otherwise, a value of 0 is returned.

Remarks

A control character is any character whose value is from 0 to 31.

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

Example

```
#include <ctype.h>
#include <stdio.h>

main ()
{
    printf ("%d %d %d \u", iscntrl ('\u'), iscntrl ('\A'),
           iscntrl ('f'));
}
```

produces the following:

1 0 1

isdigit

Tests for a decimal-digit character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>
```

```
int isdigit (
    int  c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

isdigit returns a nonzero value when the argument is a digit. Otherwise, a value of 0 is returned.

Remarks

The isdigit function or macro tests for any decimal-digit character (0 to 9).

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [isctrl \(page 31\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

Example

```
#include <stdio.h>
#include <ctype.h>

main ()
{
    char  *s, *p = "QRSTU1234ABC";
    for (s = p; *s; s++)
        printf ("%d", isdigit (*s));
}
```

produces the following:

0 0 0 0 0 1 1 1 1 0 0 0

isgraph

Tests for any printable character (except a space character) (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>
```

```
int isgraph (  
    int    c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

This function or macro returns a nonzero value when the argument is a printable character (except a space character). Otherwise, a value of 0 is returned.

Remarks

The `isgraph` function or macro tests for any printable character (except a space character). The `isprint` function is similar, except that the space character is also included in the character set being tested.

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [iscntrl \(page 31\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

islower

Tests for a lowercase character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>
```

```
int islower (
    int  c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

islower returns a nonzero value when the argument is a lowercase character. Otherwise, a value of 0 is returned.

Remarks

The islower function or macro tests for any lowercase character (a to z) in the ASCII code set.

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [isctrl \(page 31\)](#), [isdigit \(page 33\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

isprint

Tests for a printable character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>
```

```
int isprint (
    int c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

isprint returns a nonzero value when the argument is a printable character. Otherwise, a value of 0 is returned.

Remarks

The isprint function or macro tests for any printable character, including a space character.

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [iscntrl \(page 31\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

Example

```
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>

main()
{
    int i;
    while((i = getch()) != '0')
        printf("%s\r\n", isprint(i) ? "yes" : "no");
}
```

ispunct

Tests for a punctuation character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>

int ispunct (
    int c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

ispunct returns a nonzero value when the argument is a punctuation character. Otherwise, a value of 0 is returned.

Remarks

The ispunct function or macro tests for any punctuation character, such as a comma (,) or a period (.).

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [isctrl \(page 31\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

isspace

Tests for a white-space character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>
```

```
int isspace (
    int c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

isspace returns a nonzero value when the argument is one of the indicated white-space characters. Otherwise, a value of 0 is returned.

Remarks

The isspace function or macro tests for the following white-space characters:

' '	Space
\f	Formfeed
\n	Newline or line feed
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [iscntrl \(page 31\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

Example

```
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>

main()
{
    int i;
    while((i = getch()) != '0')
        printf("%s\r\n",isspace(i) ? "yes" : "no");
}
```

isupper

Tests for an uppercase character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>
```

```
int isupper (
    int c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

isupper returns a nonzero value when the argument is an uppercase character. Otherwise, a value of 0 is returned.

Remarks

The isupper function or macro tests for any uppercase character (A to Z) in the ASCII code set.

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [isctrl \(page 31\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isxdigit \(page 42\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

isxdigit

Tests for a hexadecimal-digit character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>

int isxdigit (
    int c);
```

Parameters

c

(IN) Specifies the character to be tested.

Return Values

isxdigit returns a nonzero value when the argument is a hexadecimal digit. Otherwise, a value of 0 is returned.

Remarks

The isxdigit function or macro tests for any hexadecimal-digit character. These characters include digits (0 to 9) and letters (a to f or A to F).

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [isctrl \(page 31\)](#), [isdigit \(page 33\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [tolower \(page 47\)](#), [toupper \(page 49\)](#)

mblen

Determines the number of bytes comprising the multibyte character (nonoperational in NetWare® versions 3.11 and earlier)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <stdlib.h>
```

```
int mblen (
    const char *s,
    size_t n);
```

Parameters

s

(IN) Points to the array of multibyte characters.

n

(IN) Specifies the number of bytes of the array pointed to by *s* to be examined.

Return Values

This function returns a value of 0 for any of the following conditions:

s = NULL pointer

s[0] = NULL

n = 0

Remarks

At most, *n* bytes of the array pointed to by *s* are examined.

See Also

[mbstowcs \(page 44\)](#), [mbtowc \(page 45\)](#), [wcstombs \(page 51\)](#), [wctomb \(page 52\)](#)

mbstowcs

Converts a sequence of multibyte characters into their corresponding wide-character codes and stores them in an array (nonoperational in NetWare versions 3.11 and earlier)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <stdlib.h>

size_t mbstowcs (
    wchar_t      *pwcs,
    const char   *s,
    size_t       n);
```

Parameters

pwcs

(OUT) Points to the array of wide-character codes.

s

(IN) Points to the array of multibyte characters to be converted.

n

(IN) Specifies the number of codes to be stored in the array pointed to by `pwcs`.

Return Values

`mbstowcs` returns the actual number of bytes that have been copied from the array pointed to by `s` to the array pointed to by `pwcs`. The returned value is always less than `n`.

Remarks

The `mbstowcs` function converts a sequence of multibyte characters pointed to by `s` into their corresponding wide-character codes and stores not more than `n` codes into the array pointed to by `pwcs`.

The `mbstowcs` function does not convert any multibyte characters beyond the NULL character. At most, `n` elements of the array pointed to by `pwcs` are modified.

This function is currently implemented for single-byte character coding only.

See Also

[mblen \(page 43\)](#), [mbtowc \(page 45\)](#), [wcstombs \(page 51\)](#), [wctomb \(page 52\)](#)

mbtowc

Converts a single multibyte character into the wide-character code that corresponds to that multibyte character (nonoperational in NetWare versions 3.11 and earlier)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <stdlib.h>

int mbtowc (
    wchar_t    *pwc,
    const char *s,
    size_t     n);
```

Parameters

pwc

(OUT) Points to a wide-character code.

s

(IN) Points to the array of multibyte characters to be converted.

n

(IN) Specifies the number of bytes of the array pointed to by *s* to be examined.

Return Values

This function returns a value of 0 for any of the following conditions:

```
s = NULL pointer
s [0] = NULL
n = 0
```

Remarks

The `mbtowc` function converts a single multibyte character pointed to by *s* into the wide-character code that corresponds to that multibyte character.

The code for the NULL character is zero. If the multibyte character is valid and *pwc* is not a NULL pointer, the code is stored in the object pointed to by *pwc*. At most, *n* bytes of the array pointed to by *s* are examined.

This function is currently implemented for single-byte character coding only.

See Also

[mblen \(page 43\)](#), [mbstowcs \(page 44\)](#), [wcstombs \(page 51\)](#), [wctomb \(page 52\)](#)

tolower

Converts an uppercase character to the corresponding lowercase character

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>

int tolower (
    int c);
```

Parameters

c

(IN) Specifies the character to be converted to lowercase.

Return Values

The `tolower` function returns the corresponding lowercase character when the argument is an uppercase character. Otherwise, the original character is returned.

Remarks

The `tolower` function converts an uppercase character to the corresponding lowercase character in the ASCII code set.

See Also

[isalnum \(page 26\)](#), [isalpha \(page 27\)](#), [isctrl \(page 31\)](#), [isdigit \(page 33\)](#), [isgraph \(page 35\)](#), [islower \(page 36\)](#), [isprint \(page 37\)](#), [ispunct \(page 38\)](#), [isspace \(page 39\)](#), [isupper \(page 41\)](#), [isxdigit \(page 42\)](#), [strlwr \(page 232\)](#), [strupr \(page 259\)](#), [toupper \(page 49\)](#)

Example

```
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>

main()
{
    char s[] = "THIRD QUARTER REPORT";
    int i;
    for(i=0;s[i];i++) s[i] = tolower(s[i]);
```

```
printf("%s\r\n",s);  
getch();  
}
```

toupper

Converts a lowercase character to the corresponding uppercase character

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <ctype.h>

int toupper (
    int c);
```

Parameters

c

(IN) Specifies the character to be converted to uppercase.

Return Values

The `toupper` function returns the corresponding uppercase character when the argument is a lowercase letter. Otherwise, the original character is returned.

Remarks

The `toupper` function converts a lowercase character to the corresponding uppercase character in the ASCII code set.

See Also

[isalnum](#) (page 26), [isalpha](#) (page 27), [isctrl](#) (page 31), [isdigit](#) (page 33), [isgraph](#) (page 35), [islower](#) (page 36), [isprint](#) (page 37), [ispunct](#) (page 38), [isspace](#) (page 39), [isupper](#) (page 41), [isxdigit](#) (page 42), [strlwr](#) (page 232), [strupr](#) (page 259), [tolower](#) (page 47)

Example

```
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>

main()
{
    char s[] = "third quarter report";
    int i;
    for(i=0;s[i];i++) s[i] = toupper(s[i]);
```

```
printf("%s\r\n",s);  
getch();  
}
```

wcstombs

Converts a sequence of wide-character codes from an array into a sequence of multibyte characters (nonoperational in NetWare versions 3.11 and earlier)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <stdlib.h>

size_t wcstombs (
    char          *s,
    const wchar_t *pwcs,
    size_t        n);
```

Parameters

s

(OUT) Points to the array of multibyte characters.

pwcs

(IN) Points to the array of wide-character codes to be converted.

n

(IN) Specifies the number of bytes of the array pointed to by *s* to be modified.

Return Values

This function returns the actual number of bytes that have been copied from the array pointed to by *pwcs* to the array pointed to by *s*. The returned value is always less than *n*.

Remarks

The *wcstombs* function converts a sequence of wide-character codes from the array pointed to by *pwcs* into a sequence of multibyte characters and stores them in the array pointed to by *s*. The *wcstombs* function stops if a multibyte character would exceed the limit of *n* total bytes, or if the NULL character is stored. At most, *n* bytes of the array pointed to by *s* are modified.

See Also

[mblen \(page 43\)](#), [mbstowcs \(page 44\)](#), [mbtowc \(page 45\)](#), [wctomb \(page 52\)](#)

wctomb

Determines the number of bytes required to represent the multibyte character corresponding to the specified code (nonoperational in NetWare versions 3.11 and earlier)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Character Manipulation

Syntax

```
#include <stdlib.h>

int wctomb (
    char      *s,
    wchar_t   wchar);
```

Parameters

s

(OUT) Points to the array of multibyte characters.

wchar

(IN) Specifies the wide-character code.

Return

wctomb returns a value of 0 when *s* is a NULL pointer. Otherwise, it returns a value of 1.

Remarks

The wctomb function determines the number of bytes required to represent the multibyte character corresponding to the code contained in *wchar*. If *s* is not a NULL pointer, the multibyte character representation is stored in the array pointed to by *s*. At most, MB_CUR_MAX characters are stored.

wctomb is currently implemented for single-byte character coding only.

See Also

[mblen \(page 43\)](#), [mbstowcs \(page 44\)](#), [mbtowc \(page 45\)](#), [wcstombs \(page 51\)](#)

Data Manipulation Functions

3

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

- [“bsearch” on page 54](#)
- [“IntSwap” on page 56](#)
- [“LongSwap” on page 57](#)
- [“_lrotl” on page 58](#)
- [“_lrotr” on page 59](#)
- [“NWLongSwap” on page 60](#)
- [“NWWordSwap” on page 61](#)
- [“offsetof” on page 62](#)
- [“qsort” on page 63](#)
- [“_rotl” on page 65](#)
- [“_rotr” on page 66](#)

bsearch

Performs a binary search of a sorted array

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <stdlib.h>

void *bsearch (
    const void *key,
    const void *base,
    size_t num,
    size_t width,
    int (*compar) (
        const void *,
        const void *));
```

Parameters

key

(IN) Points to the string containing the characters to be matched.

base

(IN) Points to the sorted array.

num

(IN) Specifies the number of elements in the array.

width

(IN) Specifies the size (in bytes) of each element in the array.

compar

(IN) Points to the comparison function.

Return Values

Returns a pointer to the matching member of the array, or NULL if a matching object could not be found.

Remarks

`bsearch` performs a binary search of a sorted array of elements (pointed to by the `base` parameter), for an item that matches the object pointed to by the `key` parameter.

The size of each element in the array is the number of bytes specified by the `width` parameter.

The comparison function pointed to by the `compar` parameter is called with two arguments that point to elements in the array. The comparison function will return an integer less than, equal to, or greater than zero if the first argument is less than, equal to, or greater than the second parameter.

See Also

[qsort \(page 63\)](#)

IntSwap

Reverses the 2 bytes of a short integer or short unsigned

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 3.x, 4.x, 5x

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <nwstring.h>

WORD IntSwap (
    WORD    unswappedInteger);
```

Parameters

unswappedInteger

(IN) Specifies an integer for which high and low bytes are reversed.

Return Values

Returns the byte-swapped integer.

Remarks

An application can call IntSwap to reverse the high-low order of an integer being sent or received. The 80x86 family of processors store integers in low-high order. Since some of the network functions require integers to be in high-low order, integers must be converted before they are sent or received.

See Also

[LongSwap \(page 57\)](#)

LongSwap

Reverses all 4 bytes of a long integer or long unsigned

Local Servers: nonblocking

Remote Servers: N/A

NetWare Server: 3.x, 4.x, 5.x, 6.x

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <nwstring.h>

long LongSwap (
    long    unswappedLong);
```

Parameters

unswappedLong

(IN) Specifies the long integer for which all 4 bytes are to be reversed.

Return Values

Returns the reversed long integer.

Remarks

An application can call LongSwap to reverse the high-low order of a long integer being sent or received. The 80x86 family of processors store long integers in low-high order. Since some of the network functions require long integers to be in high-low order, long integers must be converted before they are sent or after they are received.

See Also

[IntSwap \(page 56\)](#)

`_lrotl`

Rotates a long value to the left by the specified number of bits

Local Servers: nonblocking

Remote Servers: N/A

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <stdlib.h>

unsigned long _lrotl (
    unsigned long value,
    unsigned int shift);
```

Parameters

value

(IN) Specifies the value to be rotated.

shift

(IN) Specifies the number of bits by which to rotate the value.

Return Values

Returns the rotated value.

Remarks

`_lrotl` rotates the unsigned long value to the left by the number of bits specified in the `shift` parameter.

To rotate an unsigned int value, call the `_rotl` function.

See Also

[_lrotr \(page 59\)](#), [_rotl \(page 65\)](#), [_rotr \(page 66\)](#)

`_lrotr`

Rotates a long value to the right by the specified number of bits

Local Servers: nonblocking

Remote Servers: N/A

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <stdlib.h>

unsigned long _lrotr (
    unsigned long  value,
    unsigned int   shift);
```

Parameters

value

(IN) Specifies the value to be rotated.

shift

(IN) Specifies the number of bits by which to rotate the value.

Return Values

Returns the rotated value.

Remarks

`_lrotr` rotates an unsigned long value to the right by the number of bits specified in the `shift` parameter.

To rotate an unsigned int value, call the `_rotr` function.

See Also

[_lrotr \(page 58\)](#), [_rotr \(page 65\)](#), [_rotr \(page 66\)](#)

NWLongSwap

Reverses the order of the bytes on the input long (nuint32) value

NetWare Server:

Platform: NLM, Windows NT, Windows 95, Windows 98

Service: Data Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

nuint32 N_API NWLongSwap (
    nuint32 swapLong);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWLongSwap
    (swapLong : nuint32
) : nuint32;
```

Parameters

swapLong

(IN) Specifies the long (nuint32) to swap.

Return Values

Returns the swapped long.

Remarks

If the original value was 0x12345678, the return value after calling NWLongSwap will be 0x78563412.

See Also

[NWWordSwap \(page 61\)](#)

NWWordSwap

Swaps the high order byte with the low order byte

NetWare Server: 3.11, 3.12, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98

Service: Data Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

nuint16 N_API  NWWordSwap (
    nuint16  swapWord);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWWordSwap
    (swapWord : nuint16
) : nuint16;
```

Parameters

swapWord

(IN) Specifies the word (nuint16) to swap.

Return Values

Returns the swapped word.

See Also

[NWLongSwap \(page 60\)](#)

offsetof

Returns the offset of an element within a structure

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <stddef.h>
```

```
size_t offsetof (
    composite,
    name);
```

Parameters

composite

(IN) Specifies the structure or union for which to return an element offset.

name

(IN) Specifies the element in the structure for which to return the offset.

Return Values

Returns the offset of the element specified by the `name` parameter.

Remarks

`offsetof` provides a portable method to return the offset of the element specified by the `name` parameter within the structure or union specified by the `composite` parameter.

`offsetof` cannot be used to initialize data. It can only be used with executable statements.

qsort

Sorts an array of elements

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <stdlib.h>

void qsort (
    void      *base,
    size_t    num,
    size_t    width,
    int       (*compar) (
                    const void *,
                    const void *));
```

Parameters

base

(IN) Points to the array to sort.

num

(IN) Specifies the number of elements in the array.

width

(IN) Specifies the size (in bytes) of each element in the array.

compar

(IN) Points to the comparison function.

Remarks

qsort sorts an array of elements (pointed to by the `base` parameter) using Hoare's Quicksort algorithm.

The size of each element in the array is the number of bytes specified by the `width` parameter.

The comparison function pointed to by the `compar` parameter is called with two arguments that point to elements in the array. The comparison function will return an integer less than, equal to, or greater than zero if the first argument is less than, equal to, or greater than the second argument.

See Also

[bsearch \(page 54\)](#)

`_rotl`

Rotates an integer value to the left by the specified number of bits

Local Servers: nonblocking

Remote Servers: N/A

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <stdlib.h>

unsigned int _rotl (
    unsigned int  value,
    unsigned int  shift);
```

Parameters

value

(IN) Specifies the value to rotate.

shift

(IN) Specifies the number of bits by which to rotate the value.

Return Values

Returns the rotated value.

Remarks

`_rotl` rotates the unsigned int value to the left by the number of bits specified in the `shift` parameter.

To rotate an unsigned long value, call the `_lrotl` function.

See Also

[_lrotl \(page 58\)](#), [_lrotr \(page 59\)](#), [_rotr \(page 66\)](#)

`_rotr`

Rotates an integer value to the right by a specified number of bits

Local Servers: nonblocking

Remote Servers: N/A

Platform: NLM

Service: Data Manipulation

Syntax

```
#include <stdlib.h>

unsigned int _rotr (
    unsigned int  value,
    unsigned int  shift);
```

Parameters

value

(IN) Specifies the value to rotate.

shift

(IN) Specifies the number of bits by which to rotate the value.

Return Values

Returns the rotated value.

Remarks

`_rotr` rotates the unsigned int value to the right by the number of bits specified in the `shift` parameter.

To rotate an unsigned long, call the `_lrotr` function.

See Also

[_rotrl \(page 58\)](#), [_lrotr \(page 59\)](#), [_rotr \(page 65\)](#)

Library Concepts

This documentation describes Library, its functions, and features.

4.1 Unique Problems of Library NLMs

NLMs that are libraries pose some unique problems:

- If a library NLM allocates resources such as memory on behalf of another NLM, the library NLM needs to know when its clients are unloaded or otherwise terminated so that it can free any resources it allocated on the client's behalf.
- In many cases, a library NLM needs to have a different instance of data structures for each of its clients. The library needs some way of associating or getting to the proper client data structure whenever it is called by a client.
- Sometimes, a library NLM must allocate resources on a client's behalf—other times, it must allocate resources on its own behalf even though it has been called by a client.

The Library API solves these problems as follows:

- It allows a library to register a cleanup function that executes whenever one of its clients terminates.
- It allows you to save and retrieve a data area pointer on a per-client NLM basis.
- It allows you to specify which NLM resources are allocated when the library allocates resources.

NOTE: It is possible to write a library NLM without using the Library API.

4.2 Library API Example

The following example demonstrates how the Library API functions might be used in a "typical" library (Library X). In this example, Library X exports three functions:

- LoginToX
- DoTheWorkOfX
- LogoutFromX

Library X does any internal initialization before it is called by any NLM applications:

```
int main (int argc, char * argv[])
{
    /* Initialize myself */
    .
    .
    .
}
```

Library X registers itself with the NetWare® API (CLIB) using RegisterLibrary. RegisterLibrary must be called prior to other functions that require a library handle.

Library X saves its own NLM ID for later use. The library specifies which cleanup function should be called whenever a library client is terminated.

```
XNLMID = GetNLMID ();

AtUnload (LibraryXunloadFunc);    /* Define function to be
                                   executed if Library X is
                                   ever UNLOADED.*/

.
.
.
XLibHandle = RegisterLibrary (LibraryXClientCleanupFunc);
```

The library terminates its initial thread since it is not needed. (The library's functions are run by the client's thread of execution.)

```
.
.
.
ExitThread (TSR_THREAD, 0);
}
```

TSR_THREAD specifies to terminate the thread, but not to terminate the NLM, even if the last thread of the NLM is being terminated.

Whenever a client NLM wants to use Library X, the client NLM must first call LoginToX. This is not a restriction of the NetWare API, but a requirement that Library X imposes.

```
int LoginToX (int parm1, char * parm2, etc.)
{
```

In LoginToX, Library X allocates one ClientStruct for each client:

```
ClientStruct *newClientStructPtr;
newClientStructPtr = (ClientStruct *) malloc (
    sizeof (clientStruct));
if (newClientStructPtr == NULL)
    return -1;                                /* Code defined by Library X.
                                               Returned if LoginToX fails.*/
```

Library X uses one of the Library API functions, SaveDataAreaPtr, to associate the new ClientStruct with this particular client:

```
if (SaveDataAreaPtr (XLibHandle, newClientStructPtr))
    return -1;                                /* Error code defined by Library X.
                                               Returned if LoginToX fails.*/
```

LoginToX is successful, so it returns its success code:

```
return 0;
}
```

A Library X client then calls DoTheWorkOfX, which does the real work of Library X:

```
int DoTheWorkOfX (long parm1, short parm2, etc.)
```

DoTheWorkOfX calls the Library API function GetDataAreaPtr to retrieve the pointer to the ClientStruct that was saved when this client called LoginToX :

```
ClientStruct *clientStructPtr;
clientStructPtr = GetDataAreaPtr (XLibHandle);

if (clientStructPtr == NULL)
    return -2;                /* Return code defined by Library
                              X. Passed back when
                              DoTheWorkOfX is called
                              without first calling LoginToX.*/
```

DoTheWorkOfX performs the work, which includes allocating another data structure for the client. This data structure is pointed to by a field in ClientStruct, which means Library X can later get to this additional structure because it can get to ClientStruct by calling GetDataAreaPtr.

Suppose also that, during the execution of DoTheWorkOfX, an SPX™ socket must be opened to get data from some other source. Because of the way Library X is designed (Library X uses the same SPX socket for all clients, once it is opened), the open SPX socket should be considered a resource of Library X, not a resource of the client NLM. If the socket was a resource of the client, the socket would be closed when the client was terminated, which would be inappropriate with the way Library X is designed. So Library X does the following to cause the SPX socket to be counted as Library X's own resource:

```
clientNLMID = SetNLMID (XNLMID);    /* Switch to Library X.*/

if (SpxOpenSocket (&Xsocket) != 0)

    /* Do the action for failure of SpxOpen

SetNLMID (clientNLMID);            /* Switch back to client.*/
```

NOTE: Switching back to clients is very important. If this is not done, any resources the client allocates are considered Library X's resources.

After DoTheWorkOfX completes, it returns to the client. The client calls DoTheWorkOfX as many times as it needs to. When the client no longer needs to use Library X, it calls LogoutFromX. Calling LogoutFromX is part of the design of Library X, not of the NetWare API.

```
int LogoutFromX (int parm1, char * parm2, etc.)

ClientStruct *clientStructPtr;
clientStructPtr = GetDataAreaPtr (XLibHandle);
if (clientStructPtr == NULL)
    return -2;

/* Cleanup all the resources allocated for the client */
.
.
.
/* Indicate that this client NLM is no longer a client */
SaveDataAreaPtr(XLibHandle, NULL);

return 0;                /* Successfully logged out from X.*/
```

If the client of Library X was UNLOAD ed (or otherwise terminated) before it called LogoutFromX, then Library X's client cleanup function (LibraryXClientCleanupFunc) would be called. The client cleanup function receives one argument, which is the work area pointer (clientStructPtr) saved by Library X when it called SaveDataAreaPtr in LoginToX :

```
int LibraryXClientCleanupFunc (ClientStruct *clientStructPtr)
{
    /*
       Clean up all the resources allocated for the client
       Very similar to LogoutFromX
    */
    .
    .
    .
    /*
       Not necessary to call SaveDataAreaPtr (XLibHandle, NULL)
       here, because it is done automatically when a client
       terminates
    */

    return 0;          /* Zero indicates success.
                       Nonzero indicates failure.
                       This return code is ignored at present.*/
}

```

Library X can service many client NLM applications. In many cases, Library X stays loaded as long as the server is up. In some cases, Library X can be UNLOAD ed. When it is, its function, LibraryXUnloadFunc, is called because Library X called AtUnload to define an UNLOAD function:

```
void LibraryXUnloadFunc (void)
{
    /*
       Clean up any resources allocated on Library X's own behalf
       such as the SPX socket mentioned earlier
    */
    .
    .
    .
    SpxCloseSocket (XSocket);
    .
    .
    .
    /*
       Deregister Library X as a library with the
       NetWare API Library
    */

    DeregisterLibrary (XLibHandle);

    return;
}

```

This completes the example of a typical library NLM.

4.3 Library Function List

DeregisterLibrary (page 74)

Deregisters a previously registered library NLM.

GetDataAreaPtr (page 75)

Returns a saved data area pointer for the current NLM.

RegisterLibrary (page 77)

Registers an NLM as a library with NetWare® API.

SaveDataAreaPtr (page 80)

Associates a data area pointer with a client NLM.

Library Functions

5

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

- “DeregisterLibrary” on page 74
- “GetDataAreaPtr” on page 75
- “RegisterLibrary” on page 77
- “SaveDataAreaPtr” on page 80

DeregisterLibrary

Deregisters a previously registered library NLM application

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Library

Syntax

```
#include <nwlib.h>

int DeregisterLibrary (
    LONG    libraryHandle);
```

Parameters

libraryHandle

(IN) Specifies the handle returned by RegisterLibrary.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name	Description
0	(0x00)	ESUCCESS	
22	(0x16)	EBADHNDL	Invalid library.handle was passed in.

Remarks

Typically, the DeregisterLibrary function is called from the library's AtUnload function.

See Also

[RegisterLibrary \(page 77\)](#)

GetDataAreaPtr

Returns a previously saved data area pointer for the current NLM

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Library

Syntax

```
#include <nwlib.h>

void *GetDataAreaPtr (
    LONG    libraryHandle);
```

Parameters

libraryHandle

(IN) Specifies the handle returned by a call to RegisterLibrary.

Return Values

This function returns the data area pointer if successful. If a library has called SetNLMID before calling GetDataAreaPtr, this function can return NULL.

If an error occurs, it returns EFAILURE and sets `errno` to:

Value	Hex	Name	Description
22	(0x16)	EBADHNDL	Invalid library handle was passed in.

Remarks

GetDataAreaPtr returns the data area pointer saved by a previous SaveDataAreaPtr call for the current NLM (usually the NLM calling the library; however, see GetNLMID and SetNLMID) and for the specified library (specified by the `libraryHandle` parameter).

See Also

[GetNLMID](#) (*NDK: NLM Threads Management*), [SaveDataAreaPtr](#) (page 80), [SetNLMID](#) (*NDK: NLM Threads Management*)

Example

```
#include <nwlib.h>

ClientStruct    *dataAreaPtr;
```

```
LONG          libraryHandle;  
dataAreaPtr = GetDataAreaPtr (libraryHandle);
```

RegisterLibrary

Registers an NLM as a library with the NetWare® API

Local Servers: blocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Library

Syntax

```
#include <nwlib.h>

LONG RegisterLibrary (
    int (*clientCleanupFunc) (void *));
```

Parameters

clientCleanupFunc

(IN) Points to a function to be called whenever one of the clients of the library is terminated by either an UNLOAD command or because the NLM terminated itself by calling `exit`, `abort`, `ExitThread`, or so on (see "Note" below).

Return Values

This function returns a library handle or a value of `0xFFFFFFFF` if an error occurs.

Remarks

If the calling library wants to support downstream clients that are coding to LibC, call `register_library` instead of calling [RegisterLibrary \(page 77\)](#).

`RegisterLibrary` must be called prior to any other function requiring a library handle.

NOTE: The prototype of the `clientCleanupFunc` indicates that it returns a value of type `(int)`. Although NetWare currently ignores this value, `clientCleanupFunc` functions should always return `ESUCCESS` (or zero).

NOTE: It is possible to write a library NLM without using the Library functions.

For an NLM to be considered a client of a registered library by the NetWare API, the library must call `SaveDataAreaPtr` with a nonNULL data area pointer while the client NLM is the current NLM (this is usually done when the client makes its first call to the library). Only NLM applications that are clients of registered library NLM applications cause a client cleanup function to be called when they terminate.

NOTE: The library clean-up routines must be given CLIB context if they use NLM API functions that require context. You can set the context using `SetThreadGroupID` but you must set it to a thread

group ID that is part of the library, since the cleanup routine is part of the library, not part of your NLM. You should save the default thread group ID when you enter your routine and restore it, using `SetThreadGroupID`, before you leave your routine.

See Also

[DeregisterLibrary \(page 74\)](#), [SaveDataAreaPtr \(page 80\)](#)

Example of a Library NLM

```
#include <stdio.h>
#include <nwconio.h>
#include <nwlib.h>

int    LibHandle;
int    threadGroupID;

/*.....*/
int LibCleanupFunc
(
void *data
)
{
    int    curThreadGroupID;
    void   *dataAreaPtr;

    data = data;
    /*...we must establish context for the thread running
       the cleanup function...*/

    curThreadGroupID = SetThreadGroupID(threadGroupID);
    printf("Data ptr : %lX\n\n", data);
    printf("%lX Client closed.\n\n", GetThreadID());

    /*...restore the running thread's original context...*/
    SetThreadGroupID(curThreadGroupID);
    return(0);
}
/*.....*/

main()
{
    /*...save the thread group ID of the main lib thread group...*/
    threadGroupID = GetThreadGroupID();
    LibHandle = RegisterLibrary(LibCleanupFunc);
    if (LibHandle != -1)
        SuspendThread(GetThreadID());
    else
        ConsolePrintf("\n\nUnable to register library.\n\n");
}
/*.....*/
```

Example of a Client NLM

```
#include <stdio.h>
#include <nwconio.h>
#include <nwlib.h>
extern int LibHandle;
/*.....*/
main()
{
    void *dataAreaPtr;
    dataAreaPtr = (void *)0x11223344;
    /*...become a client of the library...*/
    if (SaveDataAreaPtr(LibHandle, dataAreaPtr))
        ConsolePrintf("\n\nUnable to get data area ptr for the
                        library.\n\n");
    getch();
}
/*.....*/
```

SaveDataAreaPtr

Associates a data area pointer with a particular client NLM

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Library

Syntax

```
#include <nwlib.h>

int SaveDataAreaPtr (
    LONG    libraryHandle,
    void    *dataAreaPtr);
```

Parameters

libraryHandle

(IN) Specifies the handle returned by calling RegisterLibrary.

dataAreaPtr

(IN) Points to a client NLM data area.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name	Description
0	(0x00)	ESUCCESS	
5	(0x5)	ENOMEM	Not enough memory.

Remarks

For a client NLM to be considered a client of a registered library by the NetWare API, the library must call SaveDataAreaPtr with a nonNULL data area pointer while the client is the current NLM (this is usually done when the client makes its first call to the library).

This function is normally used by library NLM applications to save a pointer to a data area the library allocates for each client. However, the dataAreaPtr parameter does not necessarily have to be a pointer, it can be an index into an array or anything else the library wants to associate with each client.

See Also

[GetDataAreaPtr](#) (page 75)

Mathematical Computation Functions

6

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

NOTE: The mathematical computation functions are exported by MATHLIB.NLM and MATHLIBC.NLM.

Use MATHLIBC if an 80387 numeric data processor is not installed.

Use MATHLIB if an 80387 numeric data processor is installed.

- [“abs” on page 85](#)
- [“acos” on page 86](#)
- [“asin” on page 87](#)
- [“atan” on page 88](#)
- [“atan2” on page 89](#)
- [“Bessel Functions” on page 90](#)
- [“cabs” on page 92](#)
- [“ceil” on page 94](#)
- [“cos” on page 95](#)
- [“cosh” on page 96](#)
- [“div” on page 97](#)
- [“exp” on page 99](#)
- [“fabs” on page 100](#)
- [“floor” on page 101](#)
- [“fmod” on page 103](#)
- [“frexp” on page 105](#)
- [“hypot” on page 107](#)
- [“labs” on page 108](#)
- [“ldexp” on page 109](#)
- [“ldiv” on page 111](#)
- [“log” on page 113](#)
- [“log10” on page 114](#)
- [“matherr” on page 115](#)
- [“max” on page 117](#)
- [“min” on page 118](#)
- [“modf” on page 119](#)

- “pow” on page 121
- “rand, rand_r” on page 123
- “RegisterMatherrHandler” on page 124
- “sin” on page 126
- “sinh” on page 127
- “sqrt” on page 128
- “srand” on page 129
- “tan” on page 130
- “tanh” on page 131

abs

Returns the absolute value of its integer argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <stdlib.h>

int abs (
    int  j);
```

Parameters

j
(IN) Specifies an integer argument.

Return Values

abs returns the absolute value of its integer argument.

Remarks

abs returns the absolute value of its integer argument *j*.

See Also

[fabs \(page 100\)](#), [labs \(page 108\)](#)

Example

```
#include <stdlib.h>
#include <stdio.h>

main ()
{
    printf ("%d %d %d\n", abs (-5), abs (0), abs (5) );
}
```

produces the following:

```
5  0  5
```

acos

Computes the principal value of the arc cosine of the specified argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double acos (
    double x);
```

Parameters

x

(IN) Specifies an argument to compute the arc cosine for.

Return Values

The `acos` function returns the arc cosine in the range $(0, \pi)$. When the argument is outside the permissible range, `errno` is set and `matherr` is called.

Remarks

A domain error occurs for arguments not in the range $(-1, 1)$.

See Also

[asin \(page 87\)](#), [atan \(page 88\)](#), [atan2 \(page 89\)](#), [matherr \(page 115\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", acos (.5) );
}
```

produces the following:

```
1.047197
```

asin

Computes the principal value of the arc sine of the specified argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double asin (
    double    x);
```

Parameters

x

(IN) Specifies an argument whose arc sine is to be computed.

Return Values

The asin function returns the arc sine in the range $(-\pi/2, \pi/2)$. When the argument is outside the permissible range, `errno` is set and `matherr` is called.

Remarks

A domain error occurs for arguments not in the range $(-1,1)$.

See Also

[acos \(page 86\)](#), [atan \(page 88\)](#), [atan2 \(page 89\)](#), [matherr \(page 115\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", asin (.5) );
}
```

produces the following:

0.523599

atan

Computes the principal value of the arc tangent of the specified argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double atan (
    double x);
```

Parameters

x

(IN) Specifies an argument whose arc tangent is to be computed.

Return Values

The atan function returns the arc tangent in the range $(-\pi/2, \pi/2)$.

See Also

[acos \(page 86\)](#), [asin \(page 87\)](#), [atan2 \(page 89\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", atan(.5) );
}
```

produces the following:

0.463648

atan2

Computes the principal value of the arc tangent of y/x , using the signs of both arguments to determine the quadrant of the return value

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>
```

```
double atan2 (
    double  y,
    double  x);
```

Parameters

x and y

(IN) Specifies the arguments whose arc tangent is to be computed.

Return Values

The `atan2` function returns the arc tangent of y/x , in the range $(-\pi, \pi)$. When the arguments are outside the permissible range, `errno` is set and `matherr` is called.

Remarks

A domain error occurs if both arguments are zero.

See Also

[acos \(page 86\)](#), [asin \(page 87\)](#), [atan \(page 88\)](#), [matherr \(page 115\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", atan2 (.5, 1.) );
}
```

produces the following:

```
0.463648
```

Bessel Functions

Returns the result of the desired Bessel function of the specified argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Mathematical Computation

Syntax

```
#include <math.h>

double j0 (
    double x);

double j1 (
    double x);

double jn (
    int n,
    double x);

double y0 (
    double x);

double y1 (
    double x);

double yn (
    int n,
    double x);
```

Return Values

The result of the desired Bessel function of the argument x is returned. For $y0$, $y1$, or yn , if x is negative the routine sets `errno` to `EDOM`, prints a `DOMAIN` error message to `stderr`, and returns `-HUGE_VAL`.

Remarks

Functions $j0$, $j1$, and jn return Bessel functions of the first kind.

Functions $y0$, $y1$, and yn return Bessel functions of the second kind. The argument x must be positive.

See Also

[matherr \(page 115\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    double x, y, z;
    x = j0( 2.4 );
    y = y1( 1.58 );
    z = jn( 3, 2.4 );
    printf( "j0(2.4) = %f, y1(1.58) = %f\n", x, y );
    printf( "jn(3,2.4) = %f\n", z );
}
```

cabs

Computes the absolute value of the complex number value by a square root calculation

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Mathematical Computation

Syntax

```
#include <math.h>

double cabs (
    struct complex value);
```

Parameters

value

(IN) Specifies the value of the complex number.

Return Values

The absolute value is returned.

Remarks

The struct complex structure is defined as

```
struct complex
{
    double real;
    double imag;
};
```

where `real` is the real part and `imag` is the imaginary part. In certain cases, overflow errors can occur that cause the `matherr` routine to be invoked.

The `cabs` function computes the absolute value of the complex number `value` by a calculation that is equivalent to

```
sqrt ((value.real * value.real) + (value.imag * value.imag))
```

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
```

```
    struct complex c = {-3.0, 4.0};  
    printf ("%f\n", cabs (c) );  
}
```

produces the following:

5.000000

ceil

Computes the smallest integer not less than x

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double ceil (
    double    x);
```

Parameters

x

(IN) Specifies an argument.

Return Values

The ceil function returns the smallest integer not less than x , expressed as a double type.

Remarks

The ceil function (ceiling function) computes the smallest integer not less than x :

```
(ceil (x) ::= -floor (-x))
```

See Also

[floor \(page 101\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f %f %f %f %f\n", ceil (-2.1), ceil (-2.),
        ceil (0.0), ceil (2.), ceil (2.1) );
}
```

produces the following:

```
-2.000000 -2.000000 0.000000 2.000000 3.000000
```

COS

Computes the cosine of the argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double cos (
    double x)
```

Parameters

x

(IN) Specifies the argument whose cosine is to be computed.

Return Values

The cos function returns the cosine value.

Remarks

The cos function computes the cosine of x (measured in radians). A large magnitude argument can yield a result with little or no significance.

See Also

[acos \(page 86\)](#), [sin \(page 126\)](#), [tan \(page 130\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    double value;
    value = cos (3.1415278);
}
```

cosh

Computes the hyperbolic cosine of the argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double cosh (
    double x);
```

Parameters

x

(IN) Specifies the argument whose hyperbolic cosine is to be computed.

Return Values

The cosh function returns the hyperbolic cosine value. When the argument is outside the permissible range, `errno` is set and `matherr` is called.

Remarks

The cosh function computes the hyperbolic cosine of x . A range error occurs if the magnitude of x is too large.

See Also

[matherr \(page 115\)](#), [sinh \(page 127\)](#), [tanh \(page 131\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", cosh (.5) );
}
```

produces the following:

```
1.127626
```

div

Calculates the quotient and remainder (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <stdlib.h>

div_t div (
    int  number,
    int  denom);
```

Parameters

numer

(IN) Specifies the numerator.

denom

(IN) Specifies the denominator.

Return Values

div returns a structure of type `div_t`, which contains the fields `quot` and `rem`.

Remarks

The `div` function or macro calculates the quotient and remainder of the division of the numerator `numer` by the denominator `denom`.

See Also

[ldiv \(page 111\)](#)

Example

```
#include <stdlib.h>
#include <stdio.h>

void print_time (int seconds)
{
    auto div_t min_sec;
    min_sec = div (seconds, 60);
    printf ("It took %d minutes and %d seconds\n",
```

```
        min_sec.quot, min_sec.rem);  
    }
```

produces the following:

It took 2 minutes and 23 seconds

exp

Computes the exponential function of the argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double exp (
    double  x);
```

Parameters

x

(IN) Specifies the argument whose exponential function is to be computed.

Return Values

The exp function returns the exponential value. When the argument is outside the permissible range, `errno` is set and `matherr` is called.

Remarks

The exp function computes the exponential function of x . A range error occurs if the magnitude of x is too large.

See Also

[log \(page 113\)](#), [matherr \(page 115\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", exp (.5) );
}
```

produces the following:

1.648721

fabs

Computes the absolute value of the argument (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double fabs (
    double    x);
```

Parameters

x

(IN) Specifies an argument whose absolute value is to be computed.

Return Values

The fabs function or macro returns the absolute value of x.

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f %f\n", fabs (.5), fabs (-.5) );
}
```

produces the following:

```
0.500000 0.500000
```

floor

Computes the largest integer not greater than the argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double floor (
    double    x);
```

Parameters

x

(IN) Specifies an argument.

Return Values

The floor function computes the largest integer not greater than x , expressed as a double type.

Remarks

The floor function computes the largest integer not greater than x .

See Also

[ceil \(page 94\)](#), [fmod \(page 103\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", floor (-3.14) );
    printf ("%f\n", floor (-3.) );
    printf ("%f\n", floor (0.) );
    printf ("%f\n", floor (3.14) );
    printf ("%f\n", floor (3.) );
}
```

produces the following:

-4.000000
-3.000000
0.000000
3.000000
3.000000

fmod

Computes the floating-point remainder of x / y

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>
```

```
double fmod (  
    double x,  
    double y);
```

Parameters

x and y

(IN) Specifies the arguments for which the floating-point remainder is to be computed.

Return Values

The `fmod` function returns the value $x - (i * y)$, for some integer i such that, if y is nonzero, the result has the same sign as x and a magnitude less than the magnitude of y . If the value of y is zero, then the value returned is zero.

Remarks

The `fmod` function computes the floating-point remainder of x / y , even if the quotient x / y is not representable.

See Also

[ceil \(page 94\)](#), [fabs \(page 100\)](#), [floor \(page 101\)](#)

Example

```
#include <math.h>  
#include <stdio.h>  
  
main ()  
{  
    printf ("%f\n", fmod (4.5, 2.0) );  
    printf ("%f\n", fmod (-4.5, 2.0) );  
    printf ("%f\n", fmod (4.5, -2.0) );  
}
```

```
    printf ("%f\n", fmod (-4.5, -2.0) );  
}
```

produces the following:

```
0.500000  
-0.500000  
0.500000  
-0.500000
```

frexp

Breaks a floating-point number into a normalized fraction and an integral power of 2

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double frexp (
    double    value,
    int       *exp);
```

Parameters

value

(IN) Specifies the original double value.

exp

(OUT) Points to the object.

Return Values

The `frexp` function returns the value of `x`, such that `x` is a double with magnitude in the interval (0.5,1) or zero, and `value` equals `x` times 2 raised to the power `*exp`. If `value` is zero, then both parts of the result are zero.

Remarks

The `frexp` function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integral power of 2 in the `int` object pointed to by `exp`.

See Also

[ldexp \(page 109\)](#), [modf \(page 119\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    int expon;
```

```
    printf ("%f %d\n", frexp (4.25, &expon), expon);  
    printf ("%f %d\n", frexp (-4.25, &expon), expon);  
}
```

produces the following:

```
0.531250 3  
-0.531250 3
```

hypot

Computes the length of the hypotenuse of a right triangle whose sides are x and y adjacent to that right angle

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Mathematical Computation

Syntax

```
#include <math.h>
```

```
double hypot (  
    double    x,  
    double    y);
```

Parameters

x and y

(IN) Specifies the sides of a right triangle adjacent to the right angle.

Return Values

The value of the hypotenuse is returned. When an error has occurred, `errno` is set.

Remarks

The `hypot` function computes the length of the hypotenuse of a right triangle whose sides are x and y adjacent to that right angle. The calculation is equivalent to

```
sqrt (x*x + y*y)
```

The computation might cause an overflow, in which case `matherr` is called.

Example

```
#include <math.h>  
#include <stdio.h>  
  
main ()  
{  
    printf ("%f\n", hypot (3.0, 4.0) );  
}
```

produces the following:

```
5.000000
```

labs

Returns the absolute value of its argument (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <stdlib.h>

long int labs (
    long int  j);
```

Parameters

j

(IN) Specifies the argument for which the absolute value is to be returned.

Return Values

The labs function or macro returns the absolute value of its argument. There is no error return.

Remarks

The labs function or macro returns the absolute value of argument j.

See Also

[abs \(page 85\)](#), [fabs \(page 100\)](#)

Example

```
#include <stdlib.h>
#include <stdio.h>

main ()
{
    long x, y;
    x = -50000L;
    y = labs (x);
}
```

ldexp

Multiplies a floating-point number by an integral power of 2

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double ldexp (
    double    x,
    int       exp);
```

Parameters

x

(IN) Specifies the floating-point number to be multiplied by an integral power of 2.

exp

(IN) Specifies the integral power of 2.

Return Values

The `ldexp` function returns the value of `x` times 2 raised to the power `exp`.

Remarks

The `ldexp` function multiplies a floating-point number by an integral power of 2. A range error can occur.

See Also

[frexp \(page 105\)](#), [modf \(page 119\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    double value;
    value=ldexp( 4.7072345, 5 );
```

```
    printf ( "%f\n", value );  
}
```

produces the following:

150.631504

ldiv

Calculates the quotient and remainder of the division of a number

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <stdlib.h>

ldiv_t ldiv (
    long int  numer,
    long int  denom);
```

Parameters

numer

(IN) Specifies the numerator.

denom

(IN) Specifies the denominator.

Return Values

This function returns a structure of type `ldiv_t` that contains the fields `quot` and `rem`, which are both of type `long int`.

Remarks

The `ldiv` function calculates the quotient and remainder of the division of the numerator `numer` by the denominator `denom`.

See Also

[div \(page 97\)](#)

Example

```
#include <stdlib.h>
#include <stdio.h>

void print_time (long int ticks);
{
    ldiv_t sec_ticks;
    ldiv_t min_sec;
```

```
sec_ticks = ldiv (ticks, 100L);
min_sec = ldiv (sec_ticks.quot, 60L);
printf ("It took %ld minutes and %ld seconds\n",
        min_sec.quot, min_sec.rem);
}
```

produces the following:

It took 14 minutes and 27 seconds

log

Computes the natural logarithm (base e) of x

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double log (
    double  x);
```

Parameters

x

(IN) Specifies the argument for which the natural logarithm is to be computed.

Return Values

The log function returns the natural logarithm of the argument. When the argument is outside the permissible range, `errno` is set and `matherr` is called. Unless `matherr` is replaced, `matherr` prints a diagnostic message using the `stderr` stream.

Remarks

A domain error occurs if the argument is negative. A range error occurs if the argument is zero.

See Also

[exp \(page 99\)](#), [log10 \(page 114\)](#), [matherr \(page 115\)](#), [pow \(page 121\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", log (.5) );
}
```

produces the following:

-0.693147

log10

Computes the logarithm (base 10) of x

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double log10 (
    double    x);
```

Parameters

x

(IN) Specifies the argument for which the logarithm is to be computed.

Return Values

The `log10` function returns the logarithm (base 10) of the argument. When an error has occurred, `errno` is set.

Remarks

The `log10` function computes the logarithm (base 10) of x . A domain error occurs if the argument is negative. A range error occurs if the argument is zero.

See Also

[exp \(page 99\)](#), [log \(page 113\)](#), [pow \(page 121\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main()
{
    printf ("%f\n", log10 (.5) );
}
```

produces the following:

```
-0.301030
```

matherr

Invoked each time an error is detected by functions in the math library

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

int matherr (
    struct exception * err_info);
```

Parameters

err_info

(OUT) Points to the exception structure that contains information about the detected error.

Return Values

The `matherr` function returns zero when an error message is to be printed; otherwise it returns a nonzero value. The default `matherr` always returns zero.

Remarks

The `matherr` function is invoked each time an error is detected by functions in the math library. The default `matherr` supplied in the library returns zero, which causes an error message to be displayed upon `stderr` and `errno` to be set with an appropriate error value. An alternative version of this function can be provided, so that mathematical errors are handled by an application.

By calling `RegisterMatherrHandler`, you can provide a developer-written version of `matherr` to take any appropriate action when an error is detected. When zero is returned, an error message is printed upon `stderr` and `errno` is set as was the case with the default function. When a nonzero value is returned, no message is printed and `errno` is not changed. The value `err_info->retval` is used as the return value for the function in which the error was detected.

The `matherr` function is passed a pointer to a structure of type `struct exception`, which contains information about the error that has been detected:

```
struct exception
{
    int         type;           /* Type of error */
    char        *name;         /* Name of function */
    double      arg1;          /* First argument to function */
    double      arg2;          /* Second argument to function */
    double      retval;        /* Default return value */
};
```

See [exception \(page 136\)](#). Only `retval` can be changed by a developer-supplied version of `matherr`.

See Also

[RegisterMatherrHandler \(page 124\)](#)

Example

```
#include <math.h>
#include <string.h>
#include <stdio.h>

/*
 * Demonstrate error routine in which negative
 * arguments to "sqrt" are treated as positive
 */
int main ()
{
    RegisterMatherrHandler (altmatherr);
    printf ("%e\n", sqrt (-5e0) );
    exit ( 0 );
}

int altmatherr (struct exception *err);
{
    if (strcmp (err->name, "sqrt") == 0)
    {
        if (err->type == DOMAIN)
        {
            err->retval = sqrt ( - (err->arg1) );
            return (1);
        }
        else
            return (0);
    }
    else
        return (0);
}
```

max

Returns the larger of two integers

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Mathematical Computation

Syntax

```
#include <stdlib.h>

int max (
    int  a,
    int  b);
```

Parameters

a

(IN) Specifies the first integer.

b

(IN) Specifies the second integer.

Return Values

The max function returns the larger of the two integers.

See Also

[min \(page 118\)](#)

Example

```
#include <stdlib.h>

external int  ComputeValue1 (void);
external int  ComputeValue2 (void);

main()
{
    int  maxValue;
    value1 = ComputeValue1 ();
    value2 = ComputeValue2 ();
    maxValue = max (value1, value2);
}
```

min

Returns the smaller of two integers

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Mathematical Computation

Syntax

```
#include <stdlib.h>

int min (
    int  a,
    int  b);
```

Parameters

a

(IN) Specifies the first integer.

b

(IN) Specifies the second integer.

Return Values

The min function returns the smaller of the two integers.

See Also

[max \(page 117\)](#)

Example

```
#include <stdlib.h>

external int ComputeValue1 (void);
external int ComputeValue2 (void);

main()
{
    int minValue;
    value1 = ComputeValue1 ();
    value2 = ComputeValue2 ();
    minValue = min (value1, value2);
}
```

modf

Breaks the argument value into integral and fractional parts

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double modf (
    double    value,
    double    *iptr);
```

Parameters

value

(IN) Specifies the value to be broken into integral and fractional parts.

iptr

(OUT) Points to the object into which the integral part is stored as a double.

Return Values

The modf function returns the signed fractional part of value.

Remarks

The modf function breaks the argument value into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a double in the object pointed to by iptr.

See Also

[frexp \(page 105\)](#), [ldexp \(page 109\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    double integral_value, fractional_part;
    fractional_part = modf (4.5, &integral_value);
```

```
printf ("%f %f\n", fractional_part, integral_value);  
fractional_part = modf (-4.5, &integral_value);  
printf ("%f %f\n", fractional_part, integral_value);  
}
```

produces the following:

```
0.500000 4.000000  
-0.500000 -4.000000
```

pow

Computes x raised to the power y

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double pow (
    double x,
    double y);
```

Parameters

x

(IN) Specifies the argument to be raised to the power y .

y

(IN) Specifies the integral power.

Return Values

The `pow` function returns the value of x raised to the power y . When the argument is outside the permissible range, `errno` is set and `matherr` is called. Unless the function is replaced, `matherr` prints a diagnostic message using the `stderr` stream.

Remarks

The `pow` function computes x raised to the power y . A domain error occurs if x is zero and y is less than or equal to 0, or if x is negative and y is not an integer. A range error can occur.

See Also

[exp \(page 99\)](#), [log \(page 113\)](#), [sqrt \(page 128\)](#)

Example

```
#include <math.h>
#include <stdio.h>
main ()
{
    printf ("%f\n", pow (1.5, 2.5) );
}
```

produces the following:

2.755676

rand, rand_r

Computes a sequence of pseudo-random integers in the range 0 to RAN_MAX (32767)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Platform: NLM

Service: Mathematical Computation

Syntax

```
#include <stdlib.h>

int rand (void);

#include <stdlib.h>

int rand_r (
    unsigned long *seed);
```

Parameters

seed

(IN/OUT) Points to a seed value (updated with each iteration).

Return Values

If the `seed` parameter is set to NULL, -1 is returned and `errno` is set to EINVAL.

If the `value` parameter is set to NULL, a value is returned but the value is not stored.

Remarks

`rand_r` is the same as `rand` except that the caller passes storage for the seed and the result rather than relying on a global variable (which has the potential of being modified by subsequent calls to `rand` by other threads in the same NLM).

`rand_r` is supported only in CLIB V 4.11 or above.

The sequence of pseudo-random integers can be started at different values by calling the `srand` function.

See Also

[srand \(page 129\)](#)

RegisterMatherrHandler

Substitutes a custom routine for matherr

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.12, 3.2, 4.x, 5.x, 6.x

Service: Mathematical Computation

Syntax

```
#include <math.h>

int RegisterMatherrHandler (
    int (*newFunc) (
        struct exception *));
```

Parameters

newFunc

(IN) Points to the routine to be used as matherr.

Return Values

RegisterMatherrHandler returns zero if successful. If a nonzero value is returned, an error occurred resulting in an error handler being registered. If an error occurs, it is most likely because the calling thread does not have CLib context.

If you have linked your program with the NWPRE.OBJ file, the RegisterMatherrHandler function returns a -1 and sets erno when an error occurs and no handler has been registered with RegisterMatherrHandler.

Remarks

RegisterMatherrHandler registers a routine (newFunc) to be called in place of matherr.

If a routine has already been registered, RegisterMatherrHandler returns a nonzero value. The following deregisters a previously registered matherr handler:

```
RegisterMatherrHandler (NULL) ;
```

The routine that is registered must meet the requirements of matherr.

See Also

[matherr \(page 115\)](#)

Example

See the example for [matherr](#) (page 115).

sin

Computes the sine of the argument (in radians)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double sin (
    double  x);
```

Parameters

x

(IN) Specifies the argument whose sine is to be computed.

Return Values

The sin function returns the sine value.

Remarks

The sin function computes the sine of x (measured in radians). A large magnitude argument can yield a result with little or no significance.

See Also

[acos \(page 86\)](#), [asin \(page 87\)](#), [atan \(page 88\)](#), [atan2 \(page 89\)](#), [cos \(page 95\)](#), [tan \(page 130\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", sin (.5) );
}
```

produces the following:

0.479426

sinh

Computes the hyperbolic sine of the specified argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double sinh (
    double    x);
```

Parameters

x

(IN) Specifies the argument whose hyperbolic sine is to be computed.

Return Values

The `sinh` function returns the hyperbolic sine value. When the argument is outside the permissible range, `errno` is set and `matherr` is called. Unless the function is replaced, `matherr` prints a diagnostic message using the `stderr` stream.

Remarks

The `sinh` function computes the hyperbolic sine of x . A range error occurs if the magnitude of x is too large.

See Also

[cosh \(page 96\)](#), [matherr \(page 115\)](#), [tanh \(page 131\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", sinh (.5) );
}
```

produces the following:

0.521095

sqrt

Computes the non-negative square root of the specified argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double sqrt (
    double x);
```

Parameters

x

(IN) Specifies the argument for which the nonnegative square root is to be computed.

Return Values

The `sqrt` function returns the value of the square root. When the argument is outside the permissible range, `errno` is set and `matherr` is called. Unless the function is replaced, `matherr` prints a diagnostic message using the `stderr` stream.

Remarks

The `sqrt` function computes the nonnegative square root of `x`. A domain error occurs if the argument is negative.

See Also

[exp \(page 99\)](#), [log \(page 113\)](#), [matherr \(page 115\)](#), [pow \(page 121\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", sqrt (.5) );
}
```

produces the following:

```
0.707107
```

srand

Starts a new sequence of pseudo-random integers to be returned by subsequent calls to the `rand` function

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Platform: NLM

Service: Mathematical Computation

Syntax

```
#include <stdlib.h>

void srand (
    unsigned int  seed);
```

Parameters

seed

(IN) Specifies the initial seed value passed in by the user.

Return Values

None

Remarks

A particular sequence of pseudo-random integers can be repeated by calling `srand` with the same `seed` parameter value.

See Also

[rand, rand_r \(page 123\)](#)

tan

Computes the tangent of the specified argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double tan (
    double  x);
```

Parameters

x

(IN) Specifies the argument whose tangent is to be computed.

Return Values

The tan function returns the tangent value. When an error has occurred, `errno` is set.

Remarks

The tan function computes the tangent of x (measured in radians). A large magnitude argument can yield a result with little or no significance.

See Also

[atan \(page 88\)](#), [atan2 \(page 89\)](#), [cos \(page 95\)](#), [sin \(page 126\)](#), [tanh \(page 131\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", tan (.5) );
}
```

produces the following:

0.546302

tanh

Computes the hyperbolic tangent of the specified argument

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <math.h>

double tanh (
    double    x);
```

Parameters

x

(IN) Specifies the argument whose hyperbolic tangent is to be computed.

Return Values

The tanh function returns the hyperbolic tangent value. When an error has occurred, `errno` is set.

Remarks

The tanh function computes the hyperbolic tangent of `x`.

When the `x` argument is large, partial or total loss of significance can occur. The `matherr` function is called in this case.

See Also

[cosh \(page 96\)](#), [sinh \(page 127\)](#), [matherr \(page 115\)](#)

Example

```
#include <math.h>
#include <stdio.h>

main ()
{
    printf ("%f\n", tanh (.5) );
}
```

produces the following:

0.462117

Mathematical Computation Structures

7

This documentation alphabetically lists the mathematical computation structures and describes their purpose, syntax, and fields.

complex

Represents a complex number

Service: Math

Defined In: math.h

Structure

```
struct complex {  
    double real ;  
    double imag ;  
};
```

Fields

real

Specifies the real part of the complex number.

imag

Specifies the imaginary part of the complex number.

div_t

Contains the results of a division in type int

Service: Math

Defined In: stdlib.h

Structure

```
typedef struct {  
    int    quot ;  
    int    rem  ;  
} div_t;
```

Fields

quot

Specifies the quotient of a division.

rem

Specifies the remainder of a division.

Remarks

Used by [div \(page 97\)](#).

exception

Contains error information used by [matherr](#) (page 115)

Service: Math

Defined In: math.h

Structure

```
struct exception
{
    int         type ;
    char        *name ;
    double      arg1 ;
    double      arg2 ;
    double      retval ;
};
```

Fields

type

Specifies the error type.

The `type` field contains one of the following values:

Value	Meaning
DOMAIN	A domain error has occurred, such as <code>sqrt(-1e0)</code> .
SING	A singularity would result, such as <code>pow(0e0,-2)</code> .
OVERFLOW	An overflow would result, such as <code>pow(10e0,100)</code> .
UNDERFLOW	An underflow would result, such as <code>pow(10e0,-100)</code> .
TLOSS	Total loss of significance would result, such as <code>exp(1000)</code> .
PLOSS	Partial loss of significance would result, such as <code>sin(10e70)</code> .

name

Points to a string containing the name of the function that detected the error.

arg1

Specifies the first argument to the function.

arg2

Specifies the second argument to the function.

retval

Specifies the default return value. This is the only field that can be changed by a developer-supplied version of `matherr`.

ldiv_t

Contains the results of a division in type long

Service: Math

Defined In: stdlib.h

Structure

```
typedef struct {  
    long    quot ;  
    long    rem  ;  
} ldiv_t;
```

Fields

quot

Specifies the quotient of a division.

rem

Specifies the remainder of a division.

Remarks

Used by [ldiv \(page 111\)](#).

Memory Allocation Functions

8

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

- [“alloca” on page 140](#)
- [“calloc” on page 142](#)
- [“free” on page 144](#)
- [“malloc” on page 145](#)
- [“_msize” on page 147](#)
- [“NWGetAllocPageOverhead” on page 148](#)
- [“NWGetAvailableMemory” on page 149](#)
- [“NWGetPageSize” on page 150](#)
- [“NWMemorySizeAddressable” on page 151](#)
- [“__qcalloc” on page 152](#)
- [“__qmalloc” on page 153](#)
- [“__qrealloc” on page 154](#)
- [“realloc” on page 155](#)
- [“stackavail” on page 157](#)

alloca

Allocates memory space for a block of memory on the stack

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

void *alloca (
    size_t    size);
```

Parameters

size

(IN) Specifies the size (in bytes) of the block of memory.

Return Values

alloca returns a pointer to the start of the allocated memory. The return value is NULL if insufficient memory is available or if the value of the `size` parameter is 0.

Remarks

Memory allocation is not limited to 64 KB. The `size` parameter is 32 bits.

The `alloca` function allocates memory space for an object of `size` bytes from the stack. The allocated space is automatically discarded when the current function exits.

The `alloca` function should not be used in an expression that is an argument to a function, because the `alloca` function manipulates the stack.

See Also

[free \(page 144\)](#), [malloc \(page 145\)](#), [NWGarbageCollect](#) (*NDK: NLM Development Concepts, Tools, and Functions*)

Example

```
#include <stdio.h>
#include <string.h>
#include <nwmalloc.h>
FILE *open_err_file (char *name)
{
    char *buffer;    /* Allocate temp buffer for file name */
```

```
buffer = alloca (strlen (name) + 5);-
if (buffer)
{
    sprintf ("buffer, %s.err", name);
    return (fopen (buffer, "w" ) );
}
return (NULL);
}
```

calloc

Allocates and clears memory space for an array of objects

Local Servers: blocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Allocation

Syntax

```
#include <stdlib.h>
#include <nwmalloc.h>
```

```
void * calloc (
    size_t    n,
    size_t    size);
```

Parameters

n

(IN) Specifies the number of objects.

size

(IN) Specifies the size (in bytes) of each object.

Return Values

calloc returns a pointer to the start of the allocated memory. The return value is NULL if insufficient memory is available or if the value of the `size` parameter is 0.

Remarks

The calloc function initializes all the memory to binary zeroes.

Memory allocation is not limited to 64 KB. The `size` parameter is 32 bits.

The calloc function calls the malloc function.

A block of memory allocated using the calloc function should be freed using the free function.

See Also

[free \(page 144\)](#), [malloc \(page 145\)](#)

Example

```
#include <nwmalloc.h>
#include <stdlib.h>
```

```
main ()
{
    int      *memoryPointer;
    size_t   n;
    memoryPointer = calloc (n, sizeof (int));
}
```

free

Frees a previously allocated memory block

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Allocation

Syntax

```
#include <stdlib.h>
#include <nwmalloc.h>

void free (
    void *ptr);
```

Parameters

ptr

(IN) Points to a memory block previously allocated by a call to `calloc`, `malloc`, or `realloc`.

Return Values

The `free` function returns no value.

Remarks

When the value of `ptr` is `NULL`, the `free` function does nothing; otherwise, the `free` function deallocates the memory block located by the `ptr` parameter. The `ptr` parameter is 32 bits.

After a call to `free`, the freed memory block is available for allocation.

See Also

[calloc \(page 142\)](#), [malloc \(page 145\)](#), [NWGarbageCollect \(NDK: NLM Development Concepts, Tools, and Functions\)](#), [realloc \(page 155\)](#)

Example

```
#include <nwmalloc.h>
#include <stdlib.h>

main ()
{
    char *ptr;
    free (ptr);
}
```

malloc

Allocates a block of memory

Local Servers: blocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>
#include <stdlib.h>

void *malloc (
    size_t  size);
```

Parameters

size

(IN) Specifies the size (in bytes) of the memory block.

Return Values

malloc returns a pointer to the start of the newly allocated memory. The return value is NULL if insufficient memory is available or if the requested size is 0.

Remarks

Memory allocation is not limited to 64 KB. The `size` parameter is 32 bits.

The malloc function blocks only if the allocated block is greater than or equal to the size of `cache_buffer+constant`.

See Also

[calloc \(page 142\)](#), [free \(page 144\)](#), [realloc \(page 155\)](#)

Example

```
#include <nwmalloc.h>
#include <stdlib.h>

main ()
{
    char      *memoryPointer;
    size_t    size;
```

```
memoryPointer = malloc (size);  
}
```

`_msize`

Returns the size of a memory block

Local Servers: blocking

Remote Servers: N/A

Classification: Other

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

size_t _msize (
    void *buffer);
```

Return Values

The `_msize` function returns the size of the memory block pointed to by `buffer`.

Remarks

The `_msize` function returns the size of the memory block pointed to by `buffer` that was allocated by a call to `calloc`, `malloc`, or `realloc`.

See Also

[calloc \(page 142\)](#), [malloc \(page 145\)](#), [realloc \(page 155\)](#)

Example

```
#include <nwmalloc.h>

main ()
{
    void *buffer;
    buffer = malloc ( 999 );
    printf ("Size of block is %u bytes\n", _msize (buffer) );
}
```

produces the following:

```
Size of block is 1000 bytes
```

The size of the memory is larger than the requested size due to space required for alignment.

NWGetAllocPageOverhead

Returns the number of bytes required to calculate integral (overhead+allocation) pages

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

size_t NWGetAllocPageOverhead (
    size_t  pageCount);
```

Parameters

pageCount

(IN) Specifies the number of pages to be allocated.

Return Values

Returns the overhead size of a memory allocation.

Remarks

NWGetAllocPageOverhead returns the number of bytes to subtract from an allocation if a caller wants to ensure that the overhead plus the allocation remain on integral pages.

See Also

[NWGetAvailableMemory \(page 149\)](#), [NWGetPageSize \(page 150\)](#), [NWMemorySizeAddressable \(page 151\)](#)

NWGetAvailableMemory

Returns the number of bytes of memory available on the server

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

LONG NWGetAvailableMemory (
    void);
```

Return Values

NWGetAvailableMemory returns the amount of memory (in bytes) available on the server.

Remarks

The amount of memory returned by NWGetAvailableMemory is the number of current cache buffers multiplied by the size of a cache buffer. The amount available will usually not be allocatable as one large chunk as it will usually not be contiguous.

See Also

[NWGetAllocPageOverhead \(page 148\)](#), [NWGetPageSize \(page 150\)](#), [NWMemorySizeAddressable \(page 151\)](#)

NWGetPageSize

Returns the page size for a given NetWare ® platform

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

size_t NWGetPageSize (
    void);
```

Return Values

Returns the page size (in bytes) for the NetWare platform on which NWGetPageSize is called.

Remarks

NWGetPageSize returns the page size of the NetWare platform on which the function is called. Currently that size is 4096 bytes, and no plans presently exist to change that size. NWGetPageSize is provided in case the NetWare page size changes at some time in the future.

See Also

[NWGetAvailableMemory \(page 149\)](#), [NWGetAllocPageOverhead \(page 148\)](#)

NWMemorySizeAddressable

Returns the amount of addressable memory

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

int NWMemorySizeAddressable (
    const void *addr,
    size_t      size);
```

Return Values

On success, NWMemorySizeAddressable returns the number of bytes of valid, addressable memory, starting at the location pointed to by the `addr` parameter. On failure, this function returns 0 and sets `errno` and `NWerrno` to `ERR_INVALID_ADDRESS` or other appropriate error code

Parameters

addr

(IN) Points to the memory to be queried.

size

(IN) Specifies the amount of memory in bytes to be marked as addressable.

Remarks

NWMemorySizeAddressable

See Also

[NWGetAvailableMemory \(page 149\)](#), [NWGetAllocPageOverhead \(page 148\)](#), [NWGetPageSize \(page 150\)](#)

__qcalloc

Allocates memory space for an array of objects

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

void *__qcalloc (
    size_t    n,
    size_t    el_size);
```

Parameters

n

(IN) Specifies the number of objects.

el_size

(IN) Specifies the size (in bytes) of the object.

Return Values

`__qcalloc` returns a pointer to the start of the allocated memory.

Remarks

The `__qcalloc` function initializes all the memory to binary zeroes. The `__qcalloc` function calls the `malloc` function.

See Also

[malloc \(page 145\)](#), [realloc \(page 155\)](#)

Example

```
#include <nwmalloc.h>

int    memoryPtr;
size_t  n;
size_t  el_size;
memoryPtr = __qcalloc (n, el_size);
```

__qmalloc

Allocates memory for an object

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>
```

```
void *__qmalloc (
    size_t  size);
```

Parameters

size

(IN) Specifies the size (in bytes) of the object.

Return Values

`__qmalloc` returns a pointer to the start of the newly allocated memory.

Remarks

The cache buffer size can be viewed or changed with the SET console command. The default cache buffer size is 4,096 bytes.

See Also

[__qcalloc \(page 152\)](#), [realloc \(page 155\)](#)

Example

```
#include <nwmalloc.h>

int      memoryPtr;
size_t   size;
memoryPtr = __qmalloc (size);
```

__qrealloc

Reallocates memory space for an object

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

void *__qrealloc (
    void      *old,
    size_t    size);
```

Parameters

old

(IN) Points to a previously allocated memory block.

size

(IN) Specifies the size (in bytes) of an object.

Return Values

`__qrealloc` returns a pointer to the start of the reallocated memory.

Remarks

The `__qrealloc` function calls the `__qmalloc` function to enlarge a block of memory.

See Also

[calloc \(page 142\)](#), [malloc \(page 145\)](#)

Example

```
#include <nwmalloc.h>

char    memoryPtr;
char    old;
size_t  size;
memoryPtr = __qrealloc (old, size);
```

realloc

Reallocates a block of memory

Local Servers: blocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>
#include <stdlib.h>
```

```
void *realloc (
    void      *oldBlk,
    size_t    size);
```

Parameters

oldBlk

(IN) Points to a previously allocated memory block.

size

(IN) Specifies the size (in bytes) of the memory block.

Return Values

realloc returns a pointer to the start of the reallocated memory. The return value is NULL if there is insufficient memory available or if the requested size is 0.

Remarks

The realloc function calls the malloc function to enlarge a block of memory.

Memory allocation is not limited to 64 KB. The size parameter is 32 bits.

When the value of the oldBlk parameter is NULL, a new block of memory of size bytes is allocated. Otherwise, the realloc function reallocates space of an object of size bytes by either:

- Shrinking the allocated size of the allocated memory block oldBlk when size is sufficiently smaller than the size of oldBlk.
- Allocating a new block and copying the contents of oldBlk to the new block when size is not sufficiently smaller than the size of oldBlk.

Because it is possible that a new block can be allocated, no other pointers should point into the memory of oldBlk. When a new block is allocated, these pointers point to freed memory, with possibly disastrous results.

See Also

[calloc \(page 142\)](#), [free \(page 144\)](#), [malloc \(page 145\)](#)

Example

```
#include <nwmalloc.h>
#include <stdlib.h>

main ()
{
    char      *memoryPointer;
    char      *oldBlk;
    size_t    size;
    memoryPointer = realloc (oldBlk, size);
}
```

stackavail

Returns the number of bytes currently available in the stack

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Memory Allocation

Syntax

```
#include <nwmalloc.h>

size_t stackavail (void);
```

Return Values

The `stackavail` function returns the number of bytes currently available in the stack.

See Also

[alloca \(page 140\)](#), [calloc \(page 142\)](#), [malloc \(page 145\)](#)

Memory Manipulation Functions

9

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

- [“memchr” on page 160](#)
- [“memcmp” on page 162](#)
- [“memcpy” on page 164](#)
- [“memicmp” on page 166](#)
- [“memmove” on page 168](#)
- [“memset” on page 170](#)

memchr

Locates the first occurrence of the specified character (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Manipulation

Syntax

```
#include <string.h>

void *memchr (
    const void *buf,
    int ch,
    size_t length);
```

Parameters

buf

(IN) Points to the object to be searched.

ch

(IN) Specifies the character to be located.

length

(IN) Specifies the number of bytes to search.

Return Values

memchr returns a pointer to the located character or NULL if the character does not occur in the object.

Remarks

The memchr function or macro locates the first occurrence of `ch` (converted to an unsigned char) in the first `length` characters of the object pointed to by `buf`.

See Also

[memcmp \(page 162\)](#), [memcpy \(page 164\)](#), [memset \(page 170\)](#)

Example

```
#include <string.h>
#include <stdio.h>
```

```
main ()
{
    char    *where;
    char    buffer[80];
    where = memchr (buffer, 'x', 6);
    if (where == NULL)
    {
        printf ("'x' not found\n");
    }
}
```

memcmp

Compares (with case sensitivity) two blocks of memory (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Manipulation

Syntax

```
#include <string.h>

int memcmp (
    const void *s1,
    const void *s2,
    size_t     length);
```

Parameters

s1

(IN) Points to the first object.

s2

(IN) Points to the second object.

length

(IN) Specifies the number of bytes to compare.

Return Values

memcmp returns an integer less than, equal to, or greater than zero, indicating that the object pointed to by s1 is less than, equal to, or greater than the object pointed to by s2.

Remarks

The memcmp function or macro compares the first length characters of the object pointed to by s1 to the object pointed to by s2.

See Also

[memchr \(page 160\)](#), [memcpy \(page 164\)](#), [memicmp \(page 166\)](#), [memset \(page 170\)](#)

Example

```
#include <string.h>
#include <stdio.h>
```

```
main ()
{
    char    buffer[80];
    if (memcmp (buffer, "Hello ", 6) < 0)
    {
        printf ("Less than\n");
    }
}
```

memcpy

Copies `length` characters from one buffer to another buffer (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Manipulation

Syntax

```
#include <string.h>

void *memcpy (
    void      *dst,
    const void *src,
    size_t    length);
```

Parameters

dst

(IN) Points to a buffer into which to copy the characters.

src

(IN) Points to a buffer containing the characters to be copied.

length

(IN) Specifies the number of characters to copy.

Return Values

`memcpy` returns `dst`.

Remarks

The `memcpy` function or macro copies `length` characters from the buffer pointed to by `src` into the buffer pointed to by `dst`. Copying of overlapping objects have undefined results. See the `memmove` function to copy objects that overlap.

See Also

[memchr \(page 160\)](#), [memcmp \(page 162\)](#), [memmove \(page 168\)](#), [memset \(page 170\)](#)

Example

```
#include <string.h>

main ()
```

```
{  
  char buffer[80];  
  memcpy (buffer, "Hello", 5);  
}
```

memicmp

Compares, with case insensitivity (uppercase and lowercase characters are equivalent), the first `length` characters of two objects

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Memory Manipulation

Syntax

```
#include <string.h>

int memicmp (
    const void    *s1,
    const void    *s2,
    size_t        length);
```

Parameters

s1

(IN) Points to the first object.

s2

(IN) Points to the second object.

length

(IN) Specifies the number of characters to compare.

Return Values

The `memicmp` function returns an integer less than, equal to, or greater than zero, indicating that the object pointed to by `s1` is less than, equal to, or greater than the object pointed to by `s2`.

Remarks

The `memicmp` function compares, with case insensitivity (uppercase and lowercase characters are equivalent), the first `length` characters of the object pointed to by `s1` to the object pointed to by `s2`.

See Also

[memchr \(page 160\)](#), [memcmp \(page 162\)](#), [memcpy \(page 164\)](#), [memset \(page 170\)](#)

Example

```
#include <string.h>
#include <stdio.h>

main ()
{
    char    buffer[80];
    if (memcmp (buffer, "Hello", 5) < 0)
    {
        printf ("Less than\n");
    }
}
```

memmove

Copies `length` characters from one buffer to another buffer

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Manipulation

Syntax

```
#include <string.h>

void *memmove (
    void      *dst,
    const void *src,
    size_t    length);
```

Parameters

dst

(IN) Points to a buffer into which to copy the characters.

src

(IN) Points to a buffer containing the characters to be copied.

length

(IN) Specifies the number of characters to move.

Return Values

The `memmove` function returns `dst`.

Remarks

The `memmove` function copies `length` characters from the buffer pointed to by `src` to the buffer pointed to by `dst`. The `dst` buffer is an exact copy of the `src` buffer. Copying of overlapping objects guarantees a buffer fill. See `memcpy` to copy objects that do not overlap.

See Also

[memcpy \(page 164\)](#), [memset \(page 170\)](#)

Example

```
#include <string.h>

main ()
```

```
{
  char buffer[80];
  memmove (buffer+1, buffer, 79);
  buffer[0] = '*';
}
```

memset

Fills the first `length` characters of an object with a specified value (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Memory Manipulation

Syntax

```
#include <string.h>

void *memset (
    void *s,
    int c,
    size_t length);
```

Parameters

s

(IN) Points to the object.

c

(IN) Specifies the value of the character.

length

(IN) Specifies the number of characters to set.

Return Values

`memset` returns `s`.

Remarks

The `memset` function or macro fills the first `length` characters of the object pointed to by `s` with the value `c`.

See Also

[memchr \(page 160\)](#), [memcmp \(page 162\)](#), [memcpy \(page 164\)](#), [memmove \(page 168\)](#)

Example

```
#include <string.h>

main ()
{
```

```
char buffer[80];  
memset (buffer, '=', 80);  
}
```


String Conversion Functions

10

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

- [“ASCIIZToLenStr” on page 174](#)
- [“ASCIIZToMaxLenStr” on page 175](#)
- [“atof” on page 177](#)
- [“atoi” on page 178](#)
- [“atol” on page 179](#)
- [“ecvt” on page 180](#)
- [“fcvt” on page 182](#)
- [“gcvt” on page 184](#)
- [“itoa” on page 186](#)
- [“LenToASCIIZStr” on page 188](#)
- [“ltoa” on page 189](#)
- [“strtod” on page 191](#)
- [“strtoi” on page 193](#)
- [“strtol” on page 195](#)
- [“strtoul” on page 197](#)
- [“ultoa” on page 199](#)
- [“utoa” on page 201](#)

ASCIIZToLenStr

Converts an ASCIIZ (NULL-terminated) string to a length-preceded string

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: String Conversion

Syntax

```
#include <nwstring.h>

int ASCIIZToLenStr (
    char  *destStr,
    char  *srcStr);
```

Parameters

destStr

(OUT) Points to the destination string.

srcStr

(IN) Points to the source string in ASCIIZ format.

Return Values

The following table lists return values and descriptions.

Value	Hex	Name	Description
0	(0x00)	ESUCCESS	
1	(0x01)	ERR_STRING_EXCEEDS_LENGTH	Longer than 255 characters.

Remarks

ASCIIZToLenStr converts an ASCIIZ (NULL-terminated) string to a length-preceded string. A length-preceded string has the length of the string in the first byte, followed by the characters of the string.

The `destStr` and `srcStr` parameters might not point at the same string.

ASCIIZToMaxLenStr

Converts an ASCIIZ (NULL-terminated) string to a length-preceded string that is not longer than the specified maximum length

Local Servers: blocking

Remote Servers: N/A

Classification: 3.12, 3.2, 4.x, 5.x, 6.x

Service: String Conversion

Syntax

```
#include <nwstring.h>

int ASCIIZToMaxLenStr (
    char *lenString,
    char *ASCIIZstring,
    int    maximumLength);
```

Parameters

lenString

(OUT) Points to the destination (length-preceded) string.

ASCIIZstring

(IN) Points to the source string in ASCIIZ format.

maximumLength

(IN) Specifies the maximum number of characters to place in the new string.

Return Values

The following table lists return values and descriptions.

Value	Name	Description
-1	EFAILURE	The ASCIIZ string was longer than the <code>mamimumLength</code> .
0	ESUCCESS	

Remarks

A length-preceded string has the length of the string in the first byte, followed by the characters in the string; it cannot exceed 255 characters. If the `ASCIIZstring` string is longer than 255 characters, this function returns `EFAILURE`, and `lenString` contains `maximumLength` characters. The remaining characters of the `ASCIIZstring` are not copied to `lenString`.

Since length-preceded strings only have one byte to store the size of the string, the maximum size of the string is 255. Passing a maximum size larger than 255 produces unpredictable results.

Example

```
#include <nwstring.h>
#include <errno.h>

main()
{
    char srcString[256];
    char destString[256];
    int ccode;
    int maxSize;
    strcpy(srcString, "This is the message");
    maxSize = 100;
    ccode = ASCIIIZToMaxLenStr(destString, srcString, maxSize);
    if(ccode == ESUCCESS)
        printf("The string fits.\n");
    else
        printf("The string was too long to fit in the allotted
              space.\n");
}
```

atof

Converts a string to double representation

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Conversion

Syntax

```
#include <stdlib.h>

double atof (
    const char *ptr);
```

Parameters

ptr

(IN) Points to the string to be converted.

Return Values

atof returns the converted value. A value of 0 is returned when the input string cannot be converted. When an error has occurred, `errno` is set

Remarks

The atof function converts the string pointed to by `ptr` to double representation.

See Also

[sscanf \(page 210\)](#), [strtod \(page 191\)](#)

Example

```
#include <stdlib.h>

main ()
{
    double x;
    x=atof("3.1415926");
}
```

atoi

Converts a string to integer representation

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Conversion

Syntax

```
#include <stdlib.h>

int atoi (
    const char *ptr);
```

Parameters

ptr

(IN) Points to the string to be converted.

Return Values

atoi returns the converted value.

Remarks

The atoi function converts the string pointed to by `ptr` to int representation.

See Also

[sscanf \(page 210\)](#), [strtoi \(page 193\)](#), [strtol \(page 195\)](#)

Example

```
#include <stdlib.h>

main ()
{
    int x;
    x=atoi("-289");
}
```

atol

Converts a string to long integer representation

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Conversion

Syntax

```
#include <stdlib.h>

long int atol (
    const char *ptr);
```

Parameters

ptr

(IN) Points to the string to be converted.

Return Values

atol returns the converted value.

Remarks

The atol function converts the string pointed to by `ptr` to long integer representation.

See Also

[sscanf \(page 210\)](#), [strtol \(page 195\)](#)

Example

```
#include <stdlib.h>

main ()
{
    long int x;
    x = atol( "-289" );
}
```

ecvt

Converts a floating-point number into a character string

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Conversion

Syntax

```
#include <stdlib.h>

char *ecvt (
    double    value,
    int       ndigits,
    int       *dec,
    int       *sign);
```

Parameters

value

(IN) Specifies the value to be converted into a string.

ndigits

(IN) Specifies the desired total number of significant digits.

dec

(OUT) Points to the decimal point position relative to the first digit.

sign

(OUT) Points to a positive or negative sign:

0 = Positive
Nonzero = Negative

Return Values

The `ecvt` function returns a pointer to a static buffer containing the converted string of digits. Note, both `ecvt` and `fcvt` use the same static buffer, except if you are using CLIB V 4.11 or above.

Remarks

The parameter `ndigits` specifies the number of significant digits desired. The converted number is rounded to `ndigits` of precision.

The character string contains only digits and is terminated by a NULL character. The integer pointed to by `dec` is filled in with a value indicating the position of the decimal point relative to the start of

the string of digits. A zero or negative value indicates that the decimal point lies to the left of the first digit. The integer pointed to by `sign` contains 0 if the number is positive, and nonzero if the number is negative.

See Also

[fcvt \(page 182\)](#), [gcvt \(page 184\)](#), [printf \(Single and Intra-File Services\)](#)

Example

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    char    str;
    int     dec, sign;
    str = ecvt( 123.456789, 6, &dec, &sign );
    printf( "str=%s,  dec=%d, sign=%d\n", str, dec, sign );
}
```

produces the following:

```
str=123457, dec=3, sign=0
```

fcvt

Converts the floating-point number value into a character string

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Conversion

Syntax

```
#include <stdlib.h>

char *fcvt (
    double   value,
    int      ndigits,
    int      *dec,
    int      *sign);
```

Parameters

value

(IN) Specifies the value to be converted into a string.

ndigits

(IN) Specifies the desired total number of significant digits to the right of the decimal point.

dec

(OUT) Points to the decimal point position relative to the first digit.

sign

(OUT) Points to a positive or negative sign: 0 = Positive; Nonzero = Negative

Return Values

The `fcvt` function returns a pointer to a static buffer containing the converted string of digits. Note, both `ecvt` and `fcvt` use the same static buffer, except if you are using CLIB V 4.11 or above.

Remarks

The difference between the `fcvt` and `ecvt` functions is that the parameter `ndigits` for the `fcvt` function specifies the number of digits desired to the right of the decimal point. The converted number is rounded to this position.

The character string contains only digits, and it is terminated by a NULL character. The integer pointed to by `dec` is filled in with a value indicating the position of the decimal point relative to the start of the string of digits. A zero or negative value indicates that the decimal point lies to the left of the first digit. The integer pointed to by `sign` contains 0 if the number is positive, and nonzero if the number is negative.

See Also

[ecvt](#) (page 180), [gcvt](#) (page 184), [printf](#) (Single and Intra-File Services)

Example

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    char *str;
    int dec, sign;
    str=fcvt( -123.456789, 5, &dec, &sign ) ;
    printf( "str=%s, dec=%d, sign=%d\n", str,dec,sign ) ;
}
```

produces the following:

```
str=12345679, dec=3, sign=-1
```

gcvt

Converts the floating-point number value into a character string and stores the result in a buffer

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Conversion

Syntax

```
#include <stdlib.h>

char *gcvt (
    double   value,
    int      ndigits,
    char     *buffer);
```

Parameters

value

(IN) Specifies the value to be converted into a string.

ndigits

(IN) Specifies the desired total number of significant digits.

buffer

(OUT) Points to the character string.

Return Values

The `gcvt` function returns a pointer to the string of digits.

Remarks

The parameter `ndigits` specifies the number of significant digits desired. The converted number are rounded to this position.

If the exponent of the number is less than -4 or is greater than or equal to the number of significant digits wanted, then the number is converted into E-format. Otherwise, the number is formatted using F-format.

See Also

[ecvt](#) (page 180), [fcvt](#) (page 182), [printf](#) (Single and Intra-File Services)

Example

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    char buffer [80] ;
    printf( "%s\n", gcvt(-123.456789, 5, buffer ) );
    printf( "%s\n", gcvt( 123.456789E+12, 5, buffer ) );
}
```

produces the following:

```
-123.46
1.2346E+014
```

itoa

Converts an integer to a string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Conversion

Syntax

```
#include <stdlib.h>

char *itoa (
    int    value,
    char  *buffer,
    int    radix);
```

Parameters

value

(IN) Specifies the integer value to be converted.

buffer

(OUT) Points to the character array.

radix

(IN) Specifies the base to be used in converting the integer.

Return Values

itoa returns the pointer to the result.

Remarks

The itoa function converts the integer value into the equivalent string in base `radix` notation, storing the result in the character array pointed to by `buffer`. A NULL character is appended to the result.

The size of `buffer` must be at least $(8 * \text{sizeof}(\text{int}) + 1)$ bytes when converting values in base 2. That makes the size 17 bytes on 16-bit machines and 33 bytes on 32-bit machines. If the value of `radix` is 10 and `value` is negative, a minus sign (-) is prepended to the result.

See Also

[atoi \(page 178\)](#), [strtol \(page 195\)](#), [utoa \(page 201\)](#)

Example

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char    buffer[20];
    int     base;
    for (base=8; base<=16; base=base+2)
        printf ("%2d %s\n", base, itoa (12765, buffer, base) );
}
```

produces the following:

```
 8 30735
10 12765
12 7479
14 491b
16 31dd
```

LenToASCIIzStr

Converts a length-preceded string to an ASCIIz (NULL-terminated)

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: String Conversion

Syntax

```
#include <nwstring.h>

int LenToASCIIzStr (
    char *destStr,
    char *srcStr);
```

Parameters

destStr

(OUT) Points to the destination string.

srcStr

(IN) Points to the source string.

Return Values

The following table lists return values and descriptions.

Value	Name	Description
0	(0x00)	ESUCCESS

The `srcStr` is only copied up to the first NULL character or the length specified, whichever is shorter. If the string is not converted (due to a NULL being encountered), the value returned is the number of characters not copied to the destination string.

Remarks

`LenToASCIIzStr` call copies and converts a length-preceded string to an ASCIIz (NULL-terminated) string.

The `srcStr` parameter is an ASCII string with the length of the string of the first byte and the actual characters of the string following it.

The `srcStr` can be the same as the `destStr`.

Itoa

Converts a long integer to a string

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Conversion

Syntax

```
#include <stdlib.h>

char *ltoa (
    long int    value,
    char        *buffer,
    int         radix);
```

Parameters

value

(IN) Specifies the integer value to be converted.

buffer

(OUT) Points to the character array.

radix

(IN) Specifies the base to be used when converting the integer.

Return Values

ltoa returns a pointer to the result.

Remarks

The ltoa function converts the integer value into the equivalent string in base `radix` notation, storing the result in the character array pointed to by `buffer`. A NULL character is appended to the result. The size of `buffer` must be at least 33 bytes when converting values in base 2. If the value of `radix` is 10 and `value` is negative, then the result is prefixed with a minus sign (-).

See Also

[atol \(page 179\)](#), [strtol \(page 195\)](#), [strtoul \(page 197\)](#), [ultoa \(page 199\)](#)

Example

```
#include <stdlib.h>
```

```
void print_value (long value)
{
    int    base;
    char   buffer[33];
    for (base=8; base<=16; base=base+2)
        printf ("%2d %s\n", base, ltoa ( value, buffer, base ) );
}
```

produces the following:

```
 8 30735
10 12765
12 7479
14 491b
16 31dd
```

strtod

Converts a string to double representation

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Conversion

Syntax

```
#include <limits.h>
#include <stdlib.h>

double strtod (
    const char *ptr,
    char **endptr);
```

Parameters

ptr

(IN) Points to the string to be converted.

endptr

(OUT) Points to the first unrecognized character.

Return Values

strtod returns the converted value. If the correct value causes overflow, plus or minus HUGE_VAL is returned according to the sign, and `errno` is set to ERANGE. If the correct value causes underflow, a value of 0 is returned, and `errno` is set to ERANGE. A value of 0 is returned when the input string cannot be converted.

Remarks

The strtod function converts the string pointed to by `ptr` to double representation. The function recognizes a string containing:

- Optional white space
- An optional plus (+) or minus (-) sign
- A sequence of digits containing an optional decimal point
- An optional e or E followed by an optionally signed sequence of digits

The conversion ends at the first unrecognized character. A pointer to that character is stored in the object to which `endptr` points if `endptr` is not NULL.

See Also

[atof \(page 177\)](#)

Example

```
#include <limits.h>
#include <stdlib.h>

double convert_pi ()
{
    double pi;
    pi = strtod ("3.141592653589793", NULL)
    return (pi);
}
```

strtoi

Converts an ASCII string to an integer

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Conversion

Syntax

```
#include <stdlib.h>

int strtoi (
    const char *str,
    int base);
```

Parameters

str

(IN) Points to the string to convert to an integer.

base

(IN) Specifies the number base of the string.

Return Values

strtoi returns the integer value for the input ASCII string and base. This value is inaccurate if a value in `str` is outside the range of the value specified in `base`. If a character in `p` is encountered outside the range of `base`, the function returns the value computed up to that point.

The `base` parameter can be 0, in which case `base` is given one of the following values:

16-if `str` begins with 0x, 0X, or contains any alphabetic characters

8-if `str` begins with 0 or 10.

See Also

[atoi \(page 178\)](#), [strtol \(page 195\)](#)

Example

```
#include <stdlib.h>

main()
{
    int decimal_value;
    int hex_value;
```

```
decimal_value = strtol( "1234567", 10 );

/* decimal_value is printed as: 1234567 */
printf( "decimal_value = %d\n", decimal_value );
decimal_value = strtol ("123a4567\n", 10);

/* decimal_value is printed as: 123 */
printf ("decimal_value = %d\n", decimal_value );
hex_value = strtol ("abc123", 16);

/* hex_value is printed as: abc123 */
printf ("hex_value = %x\n", hex_valu );
hex_value = strtol ("abch123", 16);

/* hex_value is printed as: abc */
printf ("hex_value = %x\n", hex_value);
}
```

strtol

Converts a string to an object of type long int

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Conversion

Syntax

```
#include <limits.h>
#include <stdlib.h>

long int  strtol (
    const char  *ptr,
    char       **endptr,
    int         base);
```

Parameters

ptr

(IN) Points to the string to be converted to an object.

endptr

(OUT) Points to the first unrecognized character.

base

(IN) Specifies the base of the value being converted.

Return Values

strtol returns the converted value. If the correct value would cause overflow, LONG_MAX or LONG_MIN is returned according to the sign, and errno is set to ERANGE. If base is out of range, zero is returned and errno is set to EDOM. A value of 0 is returned when the input string cannot be converted. When an error has occurred, errno is set.

Remarks

The strtol function converts the string pointed to by ptr to an object of type long int. The function recognizes a string containing:

- Optional white space
- An optional plus (+) or minus (-) sign
- A sequence of digits and letters

The conversion ends at the first unrecognized character. A pointer to that character is stored in the object to which endptr points if endptr is not NULL.

The `base` parameter must have a value between 2 and 36. The letters *a* through *z* and *A* through *Z* represent the values 10 through 35. Only those letters whose designated values are less than `base` are permitted. If the value of `base` is 16, the characters `0x` or `0X` can optionally precede the sequence of letters and digits.

See Also

[ltoa \(page 189\)](#), [strtoul \(page 197\)](#)

Example

```
#include <limits.h>
#include <stdlib.h>

main ()
{
    long int v;
    v = strtol ("12345678", NULL, 10);
}
```

strtoul

Converts a string to an unsigned long integer

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Conversion

Syntax

```
#include <limits.h>
#include <stdlib.h>

unsigned long int strtoul (
    const char *ptr,
    char **endptr,
    int base);
```

Parameters

ptr

(IN) Points to the string to be converted.

endptr

(OUT) Points to the first unrecognized character.

base

(IN) Specifies the base of the value being converted.

Return Values

strtoul returns the converted value. If the correct value would cause overflow, `ULONG_MAX` is returned and `errno` is set to `ERANGE`. If `base` is out of range, a value of 0 is returned and `errno` is set to `EDOM`. A value of 0 is returned when the input string cannot be converted. When an error has occurred, `errno` is set.

Remarks

The strtoul function converts the string pointed to by `ptr` to an unsigned long integer. The function recognizes a string containing optional white space, followed by a sequence of digits and letters. The conversion ends at the first unrecognized character. A pointer to that character is stored in the object to which `endptr` points if `endptr` is not NULL.

If `base` is zero, the first characters determine the base used for the conversion. If the first characters are `0x` or `0X`, the digits are treated as hexadecimal. If the first character is `0`, the digits are treated as octal. Otherwise, the digits are treated as decimal.

If `base` is not zero, it must have a value of between 2 and 36. The letters *a* through *z* and *A* through *Z* represent the values 10 through 35. Only those letters whose designated values are less than `base` are permitted. If the value of `base` is 16, the characters `0x` or `0X` can optionally precede the sequence of letters and digits.

See Also

[ltoa \(page 189\)](#), [strtol \(page 195\)](#), [ultoa \(page 199\)](#)

Example

```
#include <limits.h>
#include <stdlib.h>

main ()
{
    unsigned long int v;
    v = strtoul ("12345678", NULL, 10);
}
```

ultoa

Converts an unsigned long integer into the equivalent string in base notation

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Conversion

Syntax

```
#include <stdlib.h>

char *ultoa (
    unsigned long int  value,
    char              *buffer,
    int               radix);
```

Parameters

value

(IN) Specifies an unsigned long value.

buffer

(OUT) Points to the character array.

radix

(IN) Specifies the base to be used when converting the integer.

Return Values

ultoa returns the pointer to the result.

Remarks

The ultoa function converts the unsigned long integer `value` into the equivalent string in base `radix` notation, storing the result in the character array pointed to by `buffer`. A NULL character is appended to the result. The size of `buffer` must be at least 33 bytes when converting values in base 2.

See Also

[atol \(page 179\)](#), [ltoa \(page 189\)](#), [strtol \(page 195\)](#), [strtoul \(page 197\)](#), [utoa \(page 201\)](#)

Example

```
#include <stdlib.h>
```

```
void print_value (unsigned long int value)
{
    int base;
    char buffer[33];
    for (base=2; base<36; ++base)
        printf ("%s\n", ultoa (value, buffer, base) );
}
```

utoa

Converts an unsigned integer into the equivalent string in base notation

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Conversion

Syntax

```
#include <stdlib.h>

char *utoa (
    unsigned int    value,
    char            *buffer,
    int             radix);
```

Parameters

value

(IN) Specifies an unsigned integer value.

buffer

(OUT) Points to the character array.

radix

(IN) Specifies a base radix notation.

Return Values

utoa returns the pointer to the result.

Remarks

The `utoa` function converts the unsigned integer value into the equivalent string in base `radix` notation, storing the result in the character array pointed to by `buffer`. A NULL character is appended to the result. The size of `buffer` must be at least 17 bytes when converting values in base 2.

See Also

[atoi \(page 178\)](#), [itoa \(page 186\)](#), [strtol \(page 195\)](#), [strtoul \(page 197\)](#)

Example

```
#include <stdlib.h>
```

```
void print_value (unsigned int value)
{
    int base;
    char buffer[18];
    for (base=2; base<36; ++base)
        printf ("%s\n", utoa (value, buffer, base) );
}
```

String Manipulation Functions

11

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

- “LenStrCat” on page 205
- “LenStrCmp” on page 206
- “LenStrCpy” on page 207
- “sprintf” on page 208
- “sscanf” on page 210
- “strcasecmp” on page 212
- “strcat” on page 213
- “strchr” on page 215
- “strcmp” on page 217
- “strcmpi” on page 219
- “strcoll” on page 221
- “strcpy” on page 223
- “strcspn” on page 224
- “strdup” on page 226
- “strerror” on page 227
- “stricmp” on page 228
- “strlen” on page 230
- “strlist” on page 231
- “strlwr” on page 232
- “strncasecmp” on page 234
- “strncat” on page 235
- “strncmp” on page 237
- “strncpy” on page 239
- “strnicmp” on page 241
- “strnset” on page 243
- “strpbrk” on page 245
- “strrchr” on page 247
- “strev” on page 249
- “strsep” on page 250
- “strset” on page 251
- “strspn” on page 253
- “strstr” on page 255
- “strtok” on page 257

- “strupr” on page 259
- “strxfrm” on page 261
- “swab” on page 263
- “swaw” on page 265
- “vsprintf” on page 267
- “vsscanf” on page 269

LenStrCat

Concatenates two length-preceded ASCII strings

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: String Manipulation

Syntax

```
#include <nwstring.h>

char *LenStrCat (
    char *destStr,
    char *srcStr);
```

Parameters

destStr

(IN/OUT) Points to the destination string.

srcStr

(IN) Points to the source string.

Return Values

LenStrCat returns the address of the destination string.

Remarks

LenStrCat concatenates the `srcStr` onto the end of the `destStr`. The source and destination strings are two ASCII strings with the length of the string being the first byte of the string. NULL characters are copied the same as any other value. It is the programmer's job to make sure the destination string is large enough to hold the concatenated string.

LenStrCmp

Compares two length-preceded ASCII strings

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: String Manipulation

Syntax

```
#include <nwstring.h>

int LenStrCmp (
    char *string1,
    char *string2);
```

Parameters

string1

(IN) Points to the first string to be compared.

string2

(IN) Points to the second string to be compared.

Return Values

A value < 0 if string1 < string2

A value = 0 if string1 = string2

A value > 0 if string1 > string2

Remarks

LenStrCmp compares two ASCII strings preceded by a byte length. The `string1` and `string2` parameters are two ASCII strings with the length of the string being the first byte of the string. This function emulates the standard `strcmp` function but works with length-preceded strings rather than NULL-terminated strings.

LenStrCpy

Copies a length-preceded ASCII string to another string

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: String Manipulation

Syntax

```
#include <nwstring.h>

char *LenStrCpy (
    char  *destStr,
    char  *srcStr);
```

Parameters

destStr

(OUT) Points to the destination string.

srcStr

(IN) Points to the source string.

Return Values

LenStrCpy returns a pointer to `destStr`.

Remarks

The developer must ensure that the `destStr` parameter is large enough to contain the `srcStr` parameter. The source and destination strings are two ASCII strings with the length of the string being the first byte of the string. This function is similar to the standard `strcpy` function but works with length-preceded strings rather than NULL-terminated strings.

sprintf

Writes output to a specified character array under format control

Local Servers: blocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <stdio.h>

int sprintf (
    char          *buf,
    const char    *format,
    ... );
```

Parameters

buf

(OUT) Points to the character array into which to place the output.

format

(IN) Points to the format control string.

Return Values

The `sprintf` function returns the number of characters written into the array, not counting the terminating NULL character. An error can occur while converting a value for output.

Remarks

The `sprintf` function is equivalent to `fprintf`, except that the argument `buf` specifies a character array into which the generated output is placed, rather than to a file. A NULL character is placed at the end of the generated character string. The format string is described under the description for the `printf` function.

See Also

[fprintf](#), [printf](#) (Single and Intra-File Services), [vsprintf](#) (page 267)

Example

To create a temporary file name using a counter:

```
#include <stdio.h>
char *make_temp_name ()
```

```
{
    static int tempCount=0;
    static char namebuf[13];
    sprintf (namebuf, "ZZ%06d.TMP", tempCount++);
    return (namebuf);
}

main ()
{
    int i;
    for (i=0; i<3; i++)
        printf ("%s\n", make_temp_name());
}
```

sscanf

Scans input from a character string under format control

Local Servers: blocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <stdio.h>

int sscanf (
    const char    *in_string,
    const char    *format,
    ... );
```

Parameters

in_string

(IN) Points to a character string to scan.

format

(IN) Points to the format control string.

Return Values

The `sscanf` function returns EOF when scanning is terminated by reaching the end of the input string. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

Remarks

The `sscanf` function scans input from the character string `in_string` under control of the argument `format`. Following the format string is the list of addresses of items to receive values. The format string is described under the description of the `scanf` function.

See Also

[fscanf](#), [scanf](#) (Single and Intra-File Services), [vsscanf](#) (page 269)

Example

To scan the date in the form "Friday August 14 1991":

```
#include <stdio.h>
main ()
```

```
{
    int    day, year;
    char   weekday[20], month[20];
    sscanf ("Friday August 0014 1991", "%s %s %d %d",
           &weekday, &month, &day, &year);
    printf ("%s %s %d %d\n", weekday, month, day, year);
}
```

produces the following:

Friday August 14 1991

strcasecmp

Converts all characters to lowercase and then compares two strings

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

int strcasecmp (
    const char *s1,
    const char *s2);
```

Parameters

s1

(IN) Points to the first string to compare.

s2

(IN) Points to the second string to compare.

Return Values

Returns zero if the two strings match. Otherwise, it returns an integer less than or greater than zero, depending on whether `s1` is lexicographically less than or greater than `s2`.

Remarks

`strcasecmp` is identical to `strcmp`, except that the characters of each string are converted to lowercase before comparison.

Each character is treated as if it is of type `unsigned char`, and `NULL` pointers are treated as pointers to empty strings.

See Also

[strcmp \(page 217\)](#), [strcoll \(page 221\)](#), [strncasecmp \(page 234\)](#)

strcat

Appends a copy of one string to the end of a second string (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strcat (
    char      *dst,
    const char *src);
```

Parameters

dst

(OUT) Points to the string to which to append a copy of another string.

src

(IN) Points to the string to be copied.

Return Values

strcat returns the value of `dst`.

Remarks

The `strcat` function or macro appends a copy of the string pointed to by `src` (including the terminating NULL character) to the end of the string pointed to by `dst`. The first character of `src` overwrites the NULL character at the end of `dst`.

See Also

[strncat \(page 235\)](#)

Example

```
#include <string.h>

#include <stdio.h>
main ()
{
    char    buffer[80];
    strcpy (buffer, "Hello ");
```

```
    strcat (buffer, "world");  
    printf ("%s\n", buffer);  
}
```

produces the following:

Hello world

strchr

Locates the first occurrence of a specified character in a string (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strchr (
    const char *s,
    int c);
```

Parameters

s

(IN) Points to the string containing characters for which to search.

c

(IN) Specifies the character for which to search.

Return Values

strchr returns a pointer to the located character, or NULL if the character does not occur in the string.

Remarks

The strchr function or macro locates the first occurrence of *c* (converted to a char) in the string pointed to by *s*. The terminating NULL character is considered to be part of the string.

See Also

[memchr \(page 160\)](#), [strcspn \(page 224\)](#), [strrchr \(page 247\)](#), [strspn \(page 253\)](#), [strstr \(page 255\)](#), [strtok \(page 257\)](#)

Example

```
#include <string.h>
#include <stdio.h>

main ()
{
    char    buffer[80];
    char    *where;
```

```
strcpy (buffer, "01234ABCD");
where = strchr (buffer, 'x');
if (where == NULL)
{
    printf (" 'x' not found\n");
}
}
```

strcmp

Compares two strings (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

int strcmp (
    const char *s1,
    const char *s2);
```

Parameters

s1

(IN) Points to the string to be compared to the string pointed to by s2.

s2

(IN) Points to the string to be compared to the string pointed to by s1.

Return Values

strcmp returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2.

Remarks

The strcmp function or macro compares the string pointed to by s1 to the string pointed to by s2.

See Also

[stricmp \(page 228\)](#), [strncmp \(page 237\)](#), [strnicmp \(page 241\)](#)

Example

```
#include <string.h>
#include <stdio.h>

main ()
{
    printf ("%d\n", strcmp ("abcdef", "abcdef" ));
    printf ("%d\n", strcmp ("abcdef", "abc" ));
    printf ("%d\n", strcmp ("abc", "abcdef" ));
}
```

```
    printf ("%d\n", strcmp ("abcdef", "mnopqr") );  
    printf ("%d\n", strcmp ("mnopqr", "abcdef") );  
}
```

produces the following:

```
0  
1  
-1  
-1  
1
```

strcmpi

Compares, with case insensitivity, two strings (implemented for NetWare® 3.11 and above)

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>

int strcmpi (
    const char *s1,
    const char *s2);
```

Return Values

The strcmpi function returns an integer less than, equal to, or greater than 0, indicating that the string pointed to by *s1* is less than, equal to, or greater than the string pointed to by *s2*. All uppercase characters from *s1* and *s2* are mapped to lowercase characters for the purposes of doing the comparison.

Remarks

The strcmpi function compares, with case insensitivity, the string pointed to by *s1* to the string pointed to by *s2*.

See Also

[strcmp \(page 217\)](#), [stricmp \(page 228\)](#), [strncmp \(page 237\)](#), [strnicmp \(page 241\)](#)

Example

```
#include <string.h>
main()
{
    printf( "%d\n", strcmp( "AbCDEF", "abcdef" ) );
    printf( "%d\n", strcmp( "abcdef", "ABC" ) );
    printf( "%d\n", strcmp( "abc", "ABCdef" ) );
    printf( "%d\n", strcmp( "abcdef", "mnopqr" ) );
    printf( "%d\n", strcmp( "Mnopqr", "abcdef" ) );
}
```

produces the following:

```
0
100
```

-100
-12
12

strcoll

Compares two strings using the collating sequence of the current locale

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

int strcoll (
    const char *s1,
    const char *s2);
```

Parameters

s1

(IN) Points to the string to be compared to the string pointed to by s2.

s2

(IN) Points to the string to be compared to the string pointed to by s1.

Return Values

The strcoll function returns an integer less than, equal to, or greater than 0, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2, according to the collating sequence selected.

Remarks

The strcoll function compares the string pointed to by s1 to the string pointed to by s2. The comparison uses the collating sequence selected by setlocale. The function is equivalent to strcmp when the collating sequence is selected from the "C" locale.

See Also

[NWLstrcoll](#), [NWLsetlocale](#), [setlocale](#) (Internationalization), [strcmp](#) (page 217)

Example

```
#include <string.h>

main ()
{
    char    buffer[80];
```

```
    if (strcoll (buffer, "Hello") < 0)
        printf ("Less than\n");
}
```

strcpy

Copies a string into an array (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strcpy (
    char      *dst,
    const char *src);
```

Parameters

dst

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

src

(IN) Points to the string to be copied.

Return Values

strcpy returns the address of `dst`.

Remarks

The strcpy function or macro copies the string pointed to by `src` (including the terminating NULL character) into the array pointed to by `dst`. Copying of overlapping objects is not guaranteed to work properly. See the description for the memmove function to copy objects that overlap.

See Also

[memmove \(page 168\)](#), [strncpy \(page 239\)](#)

Example

```
#include <string.h>
main ()
{
    char    buffer[80];
    strcpy (buffer, "Hello ");
}
```

strcspn

Computes the length of the initial segment of a string consisting of characters not from a given set

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

size_t strcspn (
    const char *str,
    const char *charset);
```

Parameters

str

(IN) Points to the string to be scanned.

charset

(IN) Points to the set of characters for which to search.

Return Values

The `strcspn` function returns the offset position in a string where the first occurrence of `charset` begins.

Remarks

The `strcspn` function computes the length of the initial segment of the string pointed to by `str`, which consists entirely of characters not from the string pointed to by `charset`. The terminating NULL character is not considered part of `str`.

See Also

[strspn \(page 253\)](#)

Example

```
#include <string.h>

main ()
{
    printf ("%d\n", strcspn ("abcbcadef", "cba") );
    printf ("%d\n", strcspn ("xxxbcadef", "cba") );
}
```

```
    printf ("%d\n", strchr ("123456789", "cba") );  
}
```

produces the following:

```
0  
3  
9
```

strdup

Creates a duplicate of a string

Local Servers: blocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>

char *strdup (
    const char *src);
```

Parameters

src

(IN) Points to the string to be copied.

Return Values

The `strdup` function returns the pointer to the new copy of the string if successful; otherwise, it returns `NULL`.

Remarks

The `strdup` function creates a duplicate of the string pointed to by `src` and returns a pointer to the new copy. The memory for the new string is obtained by using the `malloc` function and can be freed using the `free` function.

See Also

[strcpy \(page 223\)](#)

Example

```
#include <string.h>
#include <stdio.h>

main ()
{
    char *new;
    new = strdup ("Make a copy");
    printf (new);
    free (new);
}
```

strerror

Maps an error number to an error message

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strerror (
    int  errno);
```

Parameters

errno

(IN) Specifies the error number to be mapped to an error message.

Return Values

The `strerror` function returns a pointer to the error message. The array containing the error string should not be modified by the program. This array can be overwritten by a subsequent call to the `strerror` function.

Remarks

The `strerror` function maps the error number contained in `errno` to an error message.

See Also

[perror](#)

Example

```
#include <string.h>
#include <errno.h>

main ()
{
    FILE    *fp;
    fp = fopen ("file.nam", "r");
    if (fp == NULL)
        printf ("Unable to open file: %s\n", strerror (errno) );
}
```

stricmp

Compares, with case insensitivity, two strings

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>

int stricmp (
    const char *s1,
    const char *s2);
```

Parameters

s1

(IN) Points to the string to be compared to the string pointed to by s2.

s2

(IN) Points to the string to be compared to the string pointed to by s1.

Return Values

The stricmp function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2.

Remarks

The stricmp function compares, with case insensitivity, the string pointed to by s1 to the string pointed to by s2.

See Also

[strcmp \(page 217\)](#), [strncmp \(page 237\)](#), [strnicmp \(page 241\)](#)

Example

```
#include <string.h>

main ()
{
    printf ("%d\n", stricmp ("AbCDEF", "abcdef") );
    printf ("%d\n", stricmp ("abcdef", "ABC") );
    printf ("%d\n", stricmp ("abc", "ABCdef") );
```

```
    printf ("%d\n", stricmp ("abcdef", "nopqr") );  
    printf ("%d\n", stricmp ("Mnopqr", "abcdef") );  
}
```

produces the following:

```
0  
100  
-100  
-12  
12
```

strlen

Computes the length of a string (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

size_t strlen (
    const char *s);
```

Parameters

s

(IN) Points to the string whose length is to be computed.

Return Values

strlen returns the number of characters that precede the terminating NULL character.

Remarks

The strlen function or macro computes the length of the string pointed to by s.

Example

```
#include <string.h>
#include <stdio.h>

main ()
{
    printf ("%d\n", strlen ("Howdy") );
    printf ("%d\n", strlen ("Hello world\n") );
    printf ("%d\n", strlen ("") );
}
```

produces the following:

```
5
12
0
```

strlist

Appends multiple strings together

Local Servers: nonblocking

Remote Servers: N/A

Service: String Manipulation

Syntax

```
#include <string.h>

char *strlist (
    char      *dst,
    const char *src1,
    ...);
```

Parameters

dst

(OUT) Points to the string to which to append a copy of another string.

src1 ... srcn

(IN) Points to the strings to be appended.

Return Values

strlist returns the value of `dst`.

Remarks

The `strlist` function appends multiple strings pointed to by `src1`, `src2`, ..., `srcn` to the string pointed to by `dst`. `srcn`, the last argument in the function, must be `NULL`. This function is only supported in CLIB V 4.11 or above.

strlwr

Replaces each character of a string with its lowercase equivalent

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>
char *strlwr (
    char *str);
```

Parameters

str

(IN) Points to the string to be converted to lowercase characters.

Return Values

The address of the converted string is returned.

Remarks

strlwr replaces the string `str` with lowercase characters by invoking the `tolower` function for each character in the string.

See Also

[strupr \(page 259\)](#)

Example

```
#include <string.h>
#include <stdio.h>

char source[ ] = { "A mixed-case STRING" };
main ()
{
    printf ("%s\n", source);
    printf ("%s\n", strlwr (source) );
    printf ("%s\n", source);
}
```

produces the following:

```
A mixed-case STRING  
a mixed-case string  
a mixed-case string
```

strncasecmp

Converts all characters to lowercase and then compares two strings, up to the specified number of characters

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

int strncasecmp (
    const char *s1,
    const char *s2,
    size_t     n);
```

Parameters

s1

(IN) Points to the first string to compare.

s2

(IN) Points to the second string to compare.

n

(IN) Specifies the number of characters to compare.

Return Values

Returns zero if the two strings match. Otherwise, it returns an integer less than or greater than zero, depending on whether *s1* is lexicographically less than or greater than *s2*.

Remarks

strncasecmp is identical to strncmp, except that the characters of each string are converted to lowercase before comparison.

Each character is treated as if it is of type unsigned char, and NULL pointers are treated as pointers to empty strings.

See Also

[strcasecmp \(page 212\)](#), [strncmp \(page 237\)](#)

strncat

Appends a specified number of characters of one string to another string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strncat (
    char      *dst,
    const char *src,
    size_t    n);
```

Parameters

dst

(OUT) Points to the string to which to append the characters.

src

(IN) Points to the string containing the characters to be appended to the string pointed to by `dst`.

n

(IN) Specifies the maximum number of characters to append.

Return Values

The `strncat` function returns the value of `dst`.

Remarks

The `strncat` function appends not more than `n` characters of the string pointed to by `src` to the end of the string pointed to by `dst`. The first character of `src` overwrites the NULL character at the end of `dst`. A terminating NULL character is always appended to the result.

See Also

[strcat \(page 213\)](#)

Example

```
#include <string.h>
#include <stdio.h>
```

```
char    buffer[80];
main ()
{
    strcpy (buffer, "Hello ");
    strcat (buffer, "world", 8);
    printf ("%s\n", buffer);
    strcat (buffer, "*****", 4);
    printf ("%s\n", buffer);
}
```

produces the following:

```
Hello world
Hello world****
```

strncmp

Compares a specified number of characters between two strings

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

int strncmp (
    const char *s1,
    const char *s2,
    size_t     n);
```

Parameters

s1

(IN) Points to the string to be compared to the string pointed to by s2.

s2

(IN) Points to the string to be compared to the string pointed to by s1.

n

(IN) Specifies the number of characters to be compared.

Return Values

The strncmp function returns an integer less than, equal to, or greater than 0, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2.

Remarks

The strncmp function compares not more than n characters from the string pointed to by s1 to the string pointed to by s2.

See Also

[strcmp \(page 217\)](#), [stricmp \(page 228\)](#), [strnicmp \(page 241\)](#)

Example

```
#include <string.h>
#include <stdio.h>
```

```
main ()
{
    printf ("%d\n", strncmp ("abcdef", "abcDEF", 10) );
    printf ("%d\n", strncmp ("abcdef", "abcDEF", 6) );
    printf ("%d\n", strncmp ("abcdef", "abcDEF", 3) );
    printf ("%d\n", strncmp ("abcdef", "abcDEF", 0) );
}
```

produces the following:

```
1
1
0
0
```

strncpy

Copies a specified number of characters from one string to another string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strncpy (
    char          *dst,
    const char    *src,
    size_t        n);
```

Parameters

dst

(OUT) Points to the array into which to copy the characters.

src

(IN) Points to the string containing the characters to copy.

n

(IN) Specifies the number of characters to copy.

Return Values

The `strncpy` function returns the value of `dst`.

Remarks

The `strncpy` function copies no more than `n` characters from the string pointed to by `src` into the array pointed to by `dst`. Copying of overlapping objects is not guaranteed to work properly. See the [memmove \(page 168\)](#) function if you want to copy objects that overlap.

If the string pointed to by `src` is shorter than `n` characters, NULL characters are appended to the copy in the array pointed to by `dst`, until `n` characters in all have been written. If the string pointed to by `src` is longer than `n` characters, only `n` characters are copied. No NULL characters are placed in `dst`.

See Also

[strcpy \(page 223\)](#), [strdup \(page 226\)](#)

Example

```
#include <string.h>
#include <stdio.h>

main ()
{
    char buffer[15];
    printf ("%s\n", strncpy( buffer, "abcdefg", 10) );
    printf ("%s\n", strncpy( buffer, "1234567", 6) );
    printf ("%s\n", strncpy( buffer, "abcdefg", 3) );
    printf ("%s\n", strncpy( buffer, "*****", 0) );
}
```

produces the following:

```
abcdefg
123456g
abc456g
abc456g
```

strnicmp

Compares, with case insensitivity, a specified number of characters in one string to another string

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>

int strnicmp (
    const char    *s1,
    const char    *s2,
    size_t        len);
```

Parameters

s1

(IN) Points to the string to be compared to the string pointed to by s2.

s2

(IN) Points to the string to be compared to the string pointed to by s1.

len

(IN) Specifies the number of characters to compare.

Return Values

The strnicmp function returns an integer less than, equal to, or greater than 0, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2.

Remarks

The strnicmp function compares, with case insensitivity, the string pointed to by s1 to the string pointed to by s2, for at most len characters.

See Also

[strcmp \(page 217\)](#), [stricmp \(page 228\)](#), [strncmp \(page 237\)](#)

Example

```
#include <string.h>
#include <stdio.h>
```

```
main ()
{
    printf ("%d\n", strnicmp ("abcdef", "ABCXXX", 10) );
    printf ("%d\n", strnicmp ("abcdef", "ABCXXX", 6) );
    printf ("%d\n", strnicmp ("abcdef", "ABCXXX", 3) );
    printf ("%d\n", strnicmp ("abcdef", "ABCXXX", 0) );
}
```

produces the following:

```
-20
-20
0
0
```

strnset

Sets a specified number of characters in a string to a given character

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>
```

```
char *strnset (
    char    *s1,
    int     fill,
    size_t  len);
```

Parameters

s1

(OUT) Points to the string to be filled with the specified character.

fill

(IN) Specifies the character to be copied.

len

(IN) Specifies the number of bytes into which the fill character is to be copied.

Return Values

The address of the original string `s1` is returned.

Remarks

`strnset` fills the string `s1` with the value of the argument `fill`, converted to be a character value. When the value of `len` is greater than the length of the string, the entire string is filled. Otherwise, that number of characters at the start of the string is set to the fill character.

See Also

[memset \(page 170\)](#), [strset \(page 251\)](#)

Example

```
#include <string.h>

char source[ ] = {"A sample STRING"};
```

```
main ()
{
    printf ("%s\n", source);
    printf ("%s\n", strnset (source, '=', 100) );
    printf ("%s\n", strnset (source, '*', 7) );
}
```

produces the following:

```
A sample STRING
=====
*****=====
```

strupbrk

Locates the first occurrence in one string of any character from another string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

MP Safe: Yes

Service: String Manipulation

Syntax

```
#include <string.h>

char *strupbrk (
    const char *str,
    const char *charset);
```

Parameters

str

(IN) Point to the string for which to locate the first occurrence of any character from the string pointed to by `charset`.

charset

(IN) Points to the string containing the characters to be located in the string pointed to by `str`.

Return Values

The `strupbrk` function returns a pointer to the located character or `NULL` if no character from `charset` occurs in `str`.

Remarks

The `strupbrk` function locates the first occurrence in the string pointed to by `str` of any character from the string pointed to by `charset`.

See Also

[strchr \(page 215\)](#), [strcspn \(page 224\)](#), [strrchr \(page 247\)](#), [strtok \(page 257\)](#)

Example

```
#include <string.h>

main ()
{
```

```
char *p;
p = "Find all vowels";
while (p != NULL)
{
    printf ("%s\n", p);
    p = strpbrk (p+1, "aeiouAEIOU");
}
}
```

produces the following:

```
Find all vowels
ind all vowels
all vowels
owels
els
```

strchr

Locates the last occurrence of a specified character in a string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strchr (
    const char *s,
    int c);
```

Parameters

s

(IN) Points to the string containing characters to be searched.

c

(IN) Specifies the character to locate.

Return Values

The strchr function returns a pointer to the located character or a NULL pointer if the character does not occur in the string.

Remarks

The strchr function locates the last occurrence of c (converted to a char) in the string pointed to by s. The terminating NULL character is considered to be part of the string.

See Also

[strchr \(page 215\)](#), [strpbrk \(page 245\)](#)

Example

```
#include <stdio.h>
#include <string.h>

main ()
{
    printf ("%s\n", strchr ("abcdeabcde", 'a') );
    if (strchr ("abcdeabcde", 'x') == NULL)
```

```
        printf ("NULL\n");  
    }
```

produces the following:

```
abcde  
NULL
```

strrev

Reverses the character order in a string

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>

char *strrev (
    char *s1);
```

Parameters

s1

(IN) Points to the string to be replaced.

Return Values

The address of the original string `s1` is returned.

Remarks

`strrev` replaces the string `s1` with a string whose characters are in the reverse order.

Example

```
#include <string.h>

char source[ ] = {"A sample STRING"};

main ()
{
    printf ("%s\n", source);
    printf ("%s\n", strrev (source) );
    printf ("%s\n", strrev (source) );
}
```

produces the following:

```
A sample STRING
GNIRTS elpmaS A
A sample STRING
```

strsep

Returns the specified substring

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

int strsep (
    char      *string,
    int       *which,
    const char *source);
```

Parameters

string

(OUT) Points to the returned substring.

which

(IN) Specifies the zero-based substring to return.

source

(IN) Points to the source string for which to return the specified substring.

Return Values

Returns the *n*th substring. Otherwise, returns a pointer to NULL.

Remarks

The source string must be correctly delimited by a character that does not otherwise occur in the string. The delimiter is assumed to be the first character of the source string. `strsep` will quit reading the string when it encounters NULL.

You must allocated enough memory to hold the result.

In the following string, "Tue" is the second substring:

```
:Sun:Mon:Tue:Wed:Thu:Fri:Sat:
```

See Also

[strcmp \(page 217\)](#), [strcoll \(page 221\)](#), [strncasecmp \(page 234\)](#)

strset

Fills a string with a specified character

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>

char *strset (
    char *s1,
    int fill);
```

Parameters

s1

(IN) Points to the string to be filled with the specified character.

fill

(IN) Specifies the character to be used in filling the string.

Return Values

The address of the original string `s1` is returned.

Remarks

`strset` fills the string `s1` with the character `fill`. The terminating NULL character in the original string remains unchanged.

See Also

[strnset \(page 243\)](#)

Example

```
#include <string.h>
#include <stdio.h>

char source[ ] = {"A sample STRING"};

main ()
{
    printf ("%s\n", source);
```

```
    printf ("%s\n", strstr (source, '=' ) );  
    printf ("%s\n", strstr (source, '*') );  
}
```

produces the following:

```
A sample STRING  
=====  
*****
```

strspn

Computes the length of the initial segment of a string consisting of characters from a given set

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

size_t strspn (
    const char *str,
    const char *charset);
```

Parameters

str

(IN) Points to the string for which to compute the length of the initial segment.

charset

(IN) Points to a set of characters.

Return Values

The `strspn` function returns the offset from the beginning of the string `str` where the characters in `charset` have ended.

Remarks

The `strspn` function computes the length of the initial segment of the string pointed to by `str`, which consists of characters from the string pointed to by `charset`. The terminating NULL character is not considered to be part of `charset`.

See Also

[strcspn \(page 224\)](#), [strpbrk \(page 245\)](#)

Example

```
#include <string.h>
#include <stdio.h>

main ()
{
    printf ("%d\n", strspn ("out to lunch", "aeiou") );
```

```
    printf ("%d\n", strspn ("out to lunch", "xyz") );  
}
```

produces the following:

```
2  
0
```

strstr

Scans a string for the first occurrence of a given substring

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strstr (
    const char *str,
    const char *substr);
```

Parameters

str

(IN) Points to the string to be scanned.

substr

(IN) Points to the substring for which to search.

Return Values

The `strstr` function returns a pointer to the located string, or `NULL` if the string is not found.

Remarks

The `strstr` function locates the first occurrence in the string pointed to by `str` of the sequence of characters (excluding the terminating `NULL` character) in the string pointed to by `substr`.

See Also

[strcspn \(page 224\)](#), [strpbrk \(page 245\)](#)

Example

```
#include <string.h>
#include <stdio.h>

main ()
{
    printf ("%s\n", strstr ("This is an example", "is") );
}
```

produces the following:

is is an example

strtok

Breaks a string into a sequence of tokens, each of which is delimited by a character from another string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

char *strtok (
    char      *s1,
    const char *s2);
```

Parameters

s1

(IN) Points to the string to be broken into a sequence of tokens.

s2

(IN) Points to the string containing the delimiter characters.

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 sepset, returns a pointer to the string. All of the string is considered to be a token.

Remarks

The strtok function is used to break the string pointed to by s1 into a sequence of tokens, each of which is delimited by a character from the string pointed to by s2. For example, suppose that s1 and s2 contain the following characters:

```
s1: "abcd:efgh:ijkl"
s2: ":"
```

The first call to strtok would return a pointer to a, since abcd is the first token. The second call to strtok would return a pointer to e, because efgh is the second token. The second and subsequent calls to strtok must pass a null pointer as the first argument, in order to get the next token in the string. The strtok function saves a pointer to the following character, from which the next search for a token starts when the first argument is a null pointer.

The `strtok` function replaces each delimiter that it finds with a null character, which terminates the current token. Because `strtok` can modify the original string, that string should be duplicated if the string is to be reused.

The set of delimiters used in the calls to `strtok` can be different from one call to the next.

See Also

[strcspn \(page 224\)](#), [strpbrk \(page 245\)](#)

Example

```
#include <string.h>
main ()
{
    char *p;
    char *buffer;
    char *delims = { " .," };
    buffer = strdup ("Find words, all of them.");
    printf ("%s\n", buffer);
    p = strtok (buffer, delims);
    while (p != NULL)
    {
        printf ("word: %s\n", p);
        p = strtok (NULL, delims);
    }
    printf ("%s\n", buffer );
}
```

produces the following:

```
Find words, all of them.
word: Find
word: words
word: all
word: of
word: them
Find
```

strupr

Replaces each character of a string with its uppercase equivalent

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>
```

```
char *strupr (  
    char *s1);
```

Parameters

s1

(IN) Points to the string to be replaced with uppercase characters.

Return Values

The address of the original string `s1` is returned.

Remarks

`strupr` replaces the string `s1` with uppercase characters by invoking the `toupper` function for each character in the string.

See Also

[strlwr \(page 232\)](#)

Example

```
#include <string.h>  
  
char source[ ] = { "A mixed-case STRING" };  
  
main ()  
{  
    printf ("%s\n", source);  
    printf ("%s\n", strupr ( source) );  
    printf ("%s\n", source);  
}
```

produces the following:

A mixed-case STRING
A MIXED-CASE STRING
A MIXED-CASE STRING

strxfrm

Transforms a specified number of characters from one string to another string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <string.h>

size_t strxfrm (
    char          *dst,
    const char    *src,
    size_t        n);
```

Parameters

dst

(OUT) Points to the array into which to place the transformed characters.

src

(IN) Points to the string to be transformed.

n

(IN) Specifies the number of characters to transform.

Return Values

The `strxfrm` function returns the length of the transformed string. If this length is more than `n`, the contents of the array pointed to by `dst` are indeterminate.

Remarks

The `strxfrm` function transforms, for no more than `n` characters, the string pointed to by `src` to the buffer pointed to by `dst`. The transformation uses the collating sequence selected by `setlocale` to normalize the string. If successful, the transformed string is null terminated.

The `strcoll` function normalizes two strings and then returns whether the strings are equal. It does not return the normalized strings. The `strxfrm` function allows you to normalize a string and access the results. If you pass the original string and the same string normalized with `strxfrm` to `strcoll`, `strcoll` would declare them equal.

To determine how much room is needed to store the results of the `strxfrm` function, set `dst` to `NULL` and `n` to 0 and call the function with `src` specifying the string to convert.

See Also

[setlocale](#) (Internationalization), [strcoll](#) (page 221)

swab

Copies bytes (which must be an even number of bytes) from a source buffer to a destination buffer, swapping every pair of characters

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

For CLIB V 4.11 or above:

```
#include <string.h>

void swab (
    const void *src,
    void *dst,
    size_t n);
```

For all other CLIB versions:

```
#include <string.h>

void swab (
    char *src,
    char *dst,
    int *n);
```

Parameters

src

(IN) Points to the string to be transformed.

dst

(OUT) Points to the array into which to place the transformed characters.

n

(IN) Specifies the number of characters to transform.

Return Values

None

Remarks

The `swab` function copies `n` bytes (which should be even) from `src` to `dst` swapping every pair of characters. This is useful for preparing binary data to be transferred to another machine that has a different byte ordering.

WARNING: The bytes copied must be of an even number, or the server will abend.

Example

```
#include <string.h>
#include <stdio.h>

char *msg = "hTsim seasegi  swspaep.d";
#define NBYTES 24

main()
{
    auto char buffer[80];
    printf( "%s\n", msg );
    memset( buffer, '\0', 80 );
    swab( msg, buffer, NBYTES );
    printf( "%s\n", buffer );
}
```

produces the following:

```
hTsim seasegi  swspaep.d
This message is swapped.
```

swaw

Copies words (which should be even) from a source buffer to a destination buffer, swapping every two pairs of characters

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <string.h>

void swaw (
    const void *src,
    void *dst,
    size_t n);
```

Parameters

src

(IN) Points to the string to be transformed.

dst

(OUT) Points to the array into which to place the transformed characters.

n

(IN) Specifies the number of words to transform.

Return Values

None

Remarks

The swaw function copies `n` words (which should be even) from `src` to `dst` swapping every two pairs of characters. This is useful for preparing binary data to be transferred to another machine that has a different byte ordering. This function is only supported in CLIB V 4.11 or above.

Example

```
#include <string.h>
#include <stdio.h>

char *msg = "hTsim seasegi swspaep.d";
#define NBYTES 12
```

```
main()
{
    auto char buffer[80];
    printf( "%s\n", msg );
    memset( buffer, '\0', 80 );
    swaw( msg, buffer, NBYTES );
    printf( "%s\n", buffer );
}
```

produces the following:

```
isThes mgesas iapswd.pe
This message is swapped.
```

vsprintf

Formats data under control of the format control string

Local Servers: blocking

Remote Servers: N/A

Classification: ANSI

Service: String Manipulation

Syntax

```
#include <stdarg.h>
#include <stdio.h>

int vsprintf (
    char      *buf,
    const char *format,
    va_list   arg);
```

Parameters

buf

(OUT) Points to the buffer to which to write the result.

format

(IN) Points to the format control string.

arg

(IN) Specifies a variable argument.

Return Values

The `vsprintf` function returns the number of characters written, or a negative value if an output error occurred.

Remarks

The `vsprintf` function formats data under control of the format control string and writes the result to `buf`. The format string is described under the description for `printf`. The `vsprintf` function is equivalent to `sprintf`, with the variable argument list replaced with `arg`, which has been initialized by the `va_start` macro.

See Also

[fprintf](#), [printf](#) (Single and Intra-File Services), [sprintf](#) (page 208), [va_arg](#) (page 318), [va_end](#) (page 321), [va_start](#) (page 322)

Example

This example shows the use of `vsprintf` in a general error message routine.

```
#include <stdarg.h>
#include <stdio.h>
#include <string.h>

char msgbuf[80];
char *fmtmsg (char *format, ... )

{
    va_list arglist;
    va_start (arglist, format);
    strcpy (msgbuf, "Error: ");
    vsprintf (&msgbuf[7], format, arglist);
    va_end (arglist);
    return (msgbuf);
}
```

vsscanf

Scans input from a string under format control

Local Servers: blocking

Remote Servers: N/A

Classification: Other

Service: String Manipulation

Syntax

```
#include <stdio.h>
#include <stdarg.h>

int vsscanf (
    const char    *in_string,
    const char    *format,
    va_list       arg);
```

Parameters

in_string

(IN) Points to the string to be scanned.

format

(IN) Points to the format control string.

arg

(IN) Specifies the variable argument.

Return Values

The `vsscanf` function returns EOF when the scanning is terminated by reaching the end of the input string. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

Remarks

The `vsscanf` function scans input from the string designated by `in_string` under control of the argument `format`. The format list is described with the `scanf` function.

The `vsscanf` function is equivalent to the `sscanf` function, with a variable argument list replaced with `arg`, which has been initialized using the `va_start` macro.

See Also

[fscanf](#), [scanf](#) (Single and Intra-File Services), [sscanf](#) (page 210), [va_arg](#) (page 318), [va_end](#) (page 321), [va_start](#) (page 322)

Example

```
#include <stdio.h>
#include <stdarg.h>
.
.
.
int Input (in_data, format, ...)
char *in_data, format;

{
    va_list    arglist;
    va_start (arglist, format);
    return (vsscanf (in_data, format, arglist));
}
```

Time/Date Manipulation Functions

12

This documentation alphabetically lists the time/date manipulation functions and describes their purpose, syntax, parameters, and return values.

- [“asctime, asctime_r” on page 272](#)
- [“clock” on page 274](#)
- [“_ConvertDOSTimeToCalendar” on page 275](#)
- [“_ConvertTimeToDOS” on page 277](#)
- [“ctime, ctime_r” on page 279](#)
- [“difftime” on page 281](#)
- [“GetClockStatus” on page 283](#)
- [“GetCurrentTicks” on page 285](#)
- [“gmtime, gmtime_r” on page 286](#)
- [“localtime, localtime_r” on page 288](#)
- [“ltime” on page 290](#)
- [“mktime” on page 291](#)
- [“NWPackDate” on page 293](#)
- [“NWPackDateTime” on page 295](#)
- [“NWPackTime” on page 297](#)
- [“NWUnpackDate” on page 298](#)
- [“NWUnpackDateTime” on page 300](#)
- [“NWUnpackTime” on page 302](#)
- [“SecondsToTicks” on page 303](#)
- [“strftime” on page 304](#)
- [“TicksToSeconds” on page 307](#)
- [“time” on page 308](#)
- [“tzset” on page 309](#)

asctime, asctime_r

Converts the time information into a string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

char *asctime (
    const struct tm *timeptr);

#include <time.h>

char *asctime_r (
    const struct tm *timeptr,
    char *string);
```

Parameters

timeptr

(IN) Points to the structure containing the time information to convert.

string

(OUT) Points to the converted string.

Return Values

Returns a pointer to the character string result.

Remarks

asctime and asctime_r convert the time information in the structure pointed to by `timeptr` into a string containing exactly 26 characters.

This string has the form shown in the following example:

```
Wed Mar 21 15:58:27 1990\n\0
```

All fields have a constant width. The newline character `\n` and the NULL character (`\0`) occupy the last two positions of the string. The area containing the returned string is reused each time `asctime` is called.

`asctime_r` provides the same functionality as `asctime`, but `asctime_r` is designed for use with reentrant NLMs. `asctime_r` provides that the caller pass storage for the results rather than relying on per-thread data. `asctime_r` is supported only in CLIB V 4.11 or above.

See Also

[clock](#) (page 274), [ctime](#), [ctime_r](#) (page 279), [difftime](#) (page 281), [gmtime](#), [gmtime_r](#) (page 286), [localtime](#), [localtime_r](#) (page 288), [mktime](#) (page 291), [strftime](#) (page 304), [time](#) (page 308)

Example

```
#include <stdio.h>
#include <time.h>

main ()
{
    struct tm    *time_of_day;
    time_t      ltime;
    time (&ltime);
    time_of_day = localtime (&ltime);
    printf ("Date and time is: %s.Get to work.", asctime (time_of_day));
}
```

produces the following:

```
Date and time is: Wed Mar 21 15:58:27 1990
.Get to work.
```

clock

Returns the number of hundredths of seconds since the NLM™ application began executing

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

clock_t clock (void);
```

Return Values

Returns the number of hundredths of seconds encoded into type `clock_t`.

Remarks

It is often important to run benchmarks or measure the time it takes for a process to execute. The following example shows how to call `clock` to time a process down to 1/100 of a second. In the example, assume that `CLK_TCK` is a constant equal to 100.

`asctime_r` requires the user to pass storage for the function result, rather than relying on a global variable for the result as in `asctime`.

See Also

[time \(page 308\)](#)

_ConvertDOSTimeToCalendar

Converts DOS-style date and time to calendar time

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Time/Date Manipulation

Syntax

```
#include <nwtime.h>

time_t _ConvertDOSTimeToCalendar (
    LONG    dateTime);
```

Parameters

dateTime

(IN) Specifies the DOS-style date and time to be converted.

Return Values

Returns the calendar time.

Remarks

_ConvertDOSTimeToCalendar converts DOS-style date and time used in directory entries (used in the DIR structure) to calendar time.

The standard DOS date and time format is as follows:

Figure 12-1 DOS Date and Time Format

Byte 1								Byte 0							
Year								Month				Day			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Byte 3								Byte 2							
Hour				Minute				Seconds x 2							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Calendar time represents the time in seconds since January 1, 1970 (UTC).

See [Calendar program: Example \(NDK: Sample Code\)](#).

See Also

[_ConvertTimeToDOS \(page 277\)](#)

_ConvertTimeToDOS

Converts calendar time to DOS-style date and time

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Time/Date Manipulation

Syntax

```
#include <nwdos.h>

void _ConvertTimeToDOS (
    time_t          calendarTime,
    struct _DOSDate *fileDate,
    struct _DOSTime *fileTime);
```

Parameters

calendarTime

(IN) Specifies the calendar time to be converted to DOS-style date and time.

fileDate

(OUT) Points to the DOS-style date.

fileTime

(OUT) Points to the DOS-style time.

Remarks

This function converts calendar time to DOS-style date and time used in directory entries (used in the DIR structure).

The standard DOS date and time format is as follows:

Figure 12-2 *DOS Date and Time Format*

Byte 1								Byte 0							
Year								Month				Day			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Byte 3								Byte 2							
Hour				Minute				Seconds x 2							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Calendar time represents the time in seconds since January 1, 1970 (UTC).

See Also

[_ConvertDOSTimeToCalendar \(page 275\)](#)

ctime, ctime_r

Converts the calendar time to local time in the form of a string

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

char *ctime (
    const time_t *timer);

#include <time.h>

char *ctime_r (
    const time_t *timer,
    char *string);
```

Parameters

timer

(IN) Points to the calendar time to convert to local time.

string

(OUT) Points to the converted string.

Return Values

Returns the pointer to the string containing the local time.

Remarks

ctime, ctime_r converts the calendar time information in the timer structure into a local-time string containing 26 characters.

The string has the form shown in the following example:

```
Wed Mar 21 15:58:27 1990\n\0
```

The string is equivalent to:

```
asctime (localtime (timer) )
```

All fields have a constant width. The newline character (\n) and the NULL character (\0) occupy the last two positions of the string. The area containing the returned string is reused each time ctime, ctime_r is called.

`ctime_r` provides the same functionality as `ctime`, but `ctime_r` is designed for use with reentrant NLMs. `ctime_r` provides that the caller pass storage for the results rather than relying on per-thread data. `ctime_r` is supported only in CLIB V 4.11 or above.

See Also

[asctime](#), [asctime_r](#) (page 272), [clock](#) (page 274), [difftime](#) (page 281), [gmtime](#), [gmtime_r](#) (page 286), [localtime](#), [localtime_r](#) (page 288), [mktime](#) (page 291), [strftime](#) (page 304), [time](#) (page 308)

Example

```
#include <stdio.h>
#include <time.h>

void print_time ()
{
    time_t time_of_day;
    time_of_day = time (NULL);
    printf ("It is now: %s.Get to work.\n", ctime (&time_of_day) );
}
```

produces the following:

```
It is now: Tue Dec 25 15:58:42 1990
.Get to work.
```

difftime

Calculates the difference between two calendar times (function or macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Mathematical Computation

Syntax

```
#include <time.h>

double difftime (
    time_t    time1,
    time_t    time0);
```

Parameters

time1

(IN) Specifies the calendar time to compare to `time0`.

time0

(IN) Specifies the calendar time to compare to `time1`

Return Values

`difftime` returns the difference between the two times in seconds as a double type.

Remarks

The `difftime` function or macro calculates the difference between `time1` and `time0` (`time1 - time0`).

See Also

[asctime](#), [asctime_r](#) (page 272), [clock](#) (page 274), [ctime](#), [ctime_r](#) (page 279), [gmtime](#), [gmtime_r](#) (page 286), [localtime](#), [localtime_r](#) (page 288), [mktime](#) (page 291), [strftime](#) (page 304), [time](#) (page 308)

Example

```
#include <stdio.h>
#include <time.h>

void time_test ()
{
    extern compute ();
```

```
time_t start_time, end_time;
start_time = time (NULL);
compute();
end_time = time (0);
printf("Elapsed time: %u seconds", (unsigned int)
      difftime(end_time, start_time));
}
```

produces the following:

Elapsed time: 14 seconds

GetClockStatus

Returns the time-of-day clock and status register

Local Servers: blocking

Remote Servers: N/A

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

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

void GetClockStatus (
    clockAndStatus * dataPtr);
```

Parameters

dataPtr

(OUT) Points to the clock registers and status.

Remarks

GetClockStatus succeeds whether or not time synchronization services are active. You must examine the status register to determine whether the time synchronization services are active and whether the clock is actually synchronized. There are three masks which can be used to determine the status of the clock.

CLOCK_SYNCHRONIZATION_IS_ACTIVE implies that the time synchronization NLM is loaded and active. If clock synchronization is not active, then the default state is to set CLOCK_IS_SYNCHRONIZED and clear CLOCK_IS_NETWORK_SYNCHRONIZED. If clock synchronization is not active, then the other status bits have no meaning. However, accepting the default state of CLOCK_IS_SYNCHRONIZED and taking precautions to prevent the exchange of time dependent information with other servers on the network might be acceptable to many applications.

CLOCK_IS_NETWORK_SYNCHRONIZED implies that the clock can be used for network timestamps when CLOCK_IS_SYNCHRONIZED is also true. If CLOCK_IS_NETWORK_SYNCHRONIZED is not set, then time synchronization is inactive or the server has become isolated from other servers on the network. In this case, any timestamp derived from the current clock should not be used to coordinate events with other servers.

CLOCK_IS_SYNCHRONIZED implies that the time can safely be used for timestamps for operations only on the local server. When CLOCK_IS_NETWORK_SYNCHRONIZED is also set, then timestamps can be used to coordinate events with other servers that are synchronized to network time.

The parameter type clockAndStatus is defined as

```
typedef LONG clockAndStatus[3];
```

fields of the data are:

Field	Description
dataPtr[0]	whole seconds
dataPtr[1]	fractional seconds
dataPtr[2]	status bits

Together the whole and fractional seconds fields can be interpreted as a 64-bit binary number with a binary point separating the two fields. During each clock interrupt the fractional portion is incremented by a value determined from the basic clock interrupt frequency of the host machine and by a value supplied by time synchronization services to account for drift between different clocks on the network. Attempts to use the fractional seconds as a high precision timer fail because the implied precision far exceeds the actual precision of any known hardware clock and the increment values are not guaranteed to be uniform.

The whole seconds field represents the time (in seconds) since January 1, 1970, and is reported as Universal Coordinated Time (previously known as GMT or Greenwich Mean Time) rather than local time. This format is compatible with that returned by the ANSI [time \(page 308\)](#) function.

GetCurrentTicks

Returns the current server up-time

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Time/Date Manipulation

Syntax

```
#include <nwtime.h>

LONG GetCurrentTicks (void);
```

Return Values

Returns the current server up-time in ticks (approximately eighteenthths of a second).

See Also

[clock \(page 274\)](#), [time \(page 308\)](#)

Example

```
#include <nwtime.h>

LONG serverUpTime;
serverUpTime = GetCurrentTicks ();
```

gmtime, gmtime_r

Converts calendar time into Coordinated Universal Time (UTC)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

struct tm *gmtime (
    const time_t *timer);

#include <time.h>

struct tm *gmtime_r (
    const time_t *timer,
    struct tm *timeP);
```

Parameters

timer

(IN) Points to the calendar time to convert to UTC.

timeP

(OUT) Points to the converted string.

Return Values

Returns a pointer to a structure containing the broken-down time.

Remarks

gmtime, gmtime_r converts the calendar time pointed to by the `timer` parameter into a broken-down time, expressed as UTC.

Use the NetWare® console command SET TIME to set the date and time kept by the server. Refer to the *NetWare 386 System Administration* manual for more information on this command.

The structure returned is reused every time that gmtime, gmtime_r is called.

gmtime_r provides the same functionality as gmtime, but gmtime_r is designed for use with reentrant NLMs. gmtime_r provides that the caller pass storage for the results rather than relying on per-thread data. gmtime_r is supported only in CLIB V 4.11 or above.

See Also

[asctime](#), [asctime_r](#) (page 272), [clock](#) (page 274), [ctime](#), [ctime_r](#) (page 279), [difftime](#) (page 281), [localtime](#), [localtime_r](#) (page 288), [mktime](#) (page 291), [strftime](#) (page 304), [time](#) (page 308)

localtime, localtime_r

Converts calendar time to local time

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

struct tm *localtime (
    const time_t *timer);

#include <time.h>

struct tm *localtime_r (
    const time_t *timer,
    struct tm *timeP);
```

Parameters

timer

(IN) Points to the calendar time to convert into the tm structure.

timeP

(OUT) Points to the converted string.

Return Values

Returns a pointer to the tm structure containing the time information.

Remarks

localtime, localtime_r converts the calendar time pointed to by the timer parameter into a structure of time information expressed as local time.

The calendar time (Coordinated Universal Time) is usually obtained by calling the time function.

Use the NetWare console command SET TIME to set the date and time kept by the server. Refer to the *NetWare 386 System Administration* manual for more information on this command.

The structure returned is reused every time that localtime is called.

localtime_r requires the caller to pass storage for the results rather than relying on per-thread data. localtime_r is supported only in CLIB V 4.11 or above.

`localtime_r` provides the same functionality as `localtime`, but `localtime_r` is designed for use with reentrant NLMs. `localtime_r` provides that the caller pass storage for the results rather than relying on per-thread data. `localtime_r` is supported only in CLIB V 4.11 or above.

See Also

[asctime](#), [asctime_r](#) (page 272), [clock](#) (page 274), [ctime](#), [ctime_r](#) (page 279), [difftime](#) (page 281), [gmtime](#), [gmtime_r](#) (page 286), [mktime](#) (page 291), [strftime](#) (page 304), [time](#) (page 308)

Example

```
#include <stdio.h>
#include <time.h>

void print_time ()
{
    time_t time_of_day;
    time_of_day = time (NULL);
    printf ("It is now: %s.Get to work.",asctime (localtime
        (&time_of_day) ) );
}
```

produces the following:

```
It is now: Wed Mar 21 15:58:27 1990
.Get to work.
```

ltime

Returns the current local (not UTC) time

Local Servers: nonblocking

Remote Servers: N/A

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

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

time_t ltime (
    time_t *timer);
```

Parameters

timer

(OUT) Points to the object into which the encoded local time—whose type is identical to that of `time`—has been placed. If `timer` is not NULL, the current local time is also stored in the object pointed to by this parameter.

Return Values

Returns the current *local* time encoded into the `time_t` structure.

Remarks

`ltime` is the local equivalent of `time` and includes adjustments for daylight-savings time (if it is implemented and active for the current locale). The time is encoded and represents the time in seconds since January 1, 1970 at 00:00:00. Note that a corrupted calendar time is returned, since `time_t` holds the calendar or UTC time. `ltime` returns the local time expressed in a scalar that can be stored; however, this value should not be passed to any other time function.

To retrieve the offset, in seconds, for the current locale from the UTC time and have it adjusted for daylight-savings time, use the following:

```
offset = difftime(ltime(NULL), time(NULL));
```

See Also

[difftime \(page 281\)](#), [time \(page 308\)](#)

mktime

Converts the time information in a structure into calendar time

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

time_t mktime (
    struct tm *timeptr);
```

Parameters

timeptr

(IN) Points to the structure containing the time information to convert.

Return Values

Returns the converted calendar time. On error, -1 will be returned cast as type (time_t). To test for an error condition, compare the return value to (time_t) -1.

Remarks

mktime converts the time information in the structure pointed to by the timeptr parameter into a calendar time with the same encoding used by the time function.

See Also

[asctime](#), [asctime_r](#) (page 272), [clock](#) (page 274), [ctime](#), [ctime_r](#) (page 279), [difftime](#) (page 281), [gmtime](#), [gmtime_r](#) (page 286), [localtime](#), [localtime_r](#) (page 288), [strftime](#) (page 304), [time](#) (page 308)

Example

```
#include <stdio.h>
#include <time.h>

static const char *week_day[ ] =
{
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"
};
```

```
main ()
{
    struct tm new_year;
    new_year.tm_year = 2001 - 1900;
    new_year.tm_mon  = 0;
    new_year.tm_mday = 1;
    new_year.tm_hour = 0;
    new_year.tm_min  = 0;
    new_year.tm_sec  = 0;
    new_year.tm_isdst = 0;
    mktime (&new_year);
    printf ("The next century begins on a %s\n",
           week_day [new_year.tm_wday] );
}
```

produces the following:

The next century begins on a Monday

NWPackDate

Packs a date into an nuint16

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98

Service: Time/Date Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY (nuint16) NWPackDate (
    const NW_DATE N_FAR *sDate);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWPackDate
    (const sDate : pNW_DATE
) : nuint16;
```

Parameters

sDate

(IN) Points to the NW_DATE structure.

Return Values

Returns the packed date upon successful completion.

Remarks

Many functions return dates in a packed format identical to that defined by DOS. The bits are defined as follows:

```
0-4   day
5-8   month
9-15  year minus 1980
```

NWPackDate does no validity checking on the passed information. The programmer should ensure dates are valid before passing them to NWPackDate.

See Also

[NWPackDateTime \(page 295\)](#), [NWPackTime \(page 297\)](#), [NWUnpackDate \(page 298\)](#),
[NWUnpackDateTime \(page 300\)](#), [NWUnpackTime \(page 302\)](#)

NWPackDateTime

Packs a date and time into a uint32

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98

Service: Time/Date Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

uint32 N_API NWPackDateTime (
    const NW_DATE N_FAR *sDate,
    const NW_TIME N_FAR *sTime);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWPackDateTime
    (const sDate : pNW_DATE;
    const sTime : pNW_TIME
) : uint32;
```

Parameters

sDate

(IN) Points to the NW_DATE structure (optional).

sTime

(IN) Points to the NW_TIME structure (optional).

Return Values

Returns the packed date and time upon successful completion.

Remarks

NWPackDateTime returns the packed date and time. If a parameter is NULL, the associated bits will be set to zero.

Many functions return dates in a packed format identical to that defined by DOS. Time occupies the low order word and date occupies the high order word. The bits are defined as follows:

0-4	seconds divided by two
5-10	minutes
11-15	hours (0-23)

16-20 day
21-24 month
25-31 year minus 1980

NWPackDateTime does no validity checking on the passed information. The programmer should ensure dates and/or times in the associated structures are valid before passing them to NWPackDateTime.

See [timedate.c \(../../samplecode/clib_sample/misc_timedate/timedate.c.html\)](http://.../samplecode/clib_sample/misc_timedate/timedate.c.html) for sample code.

See Also

[NWPackDate \(page 293\)](#), [NWPackTime \(page 297\)](#), [NWUnpackDate \(page 298\)](#),
[NWUnpackDateTime \(page 300\)](#), [NWUnpackTime \(page 302\)](#)

NWPackTime

Packs a time into an nuint16

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98

Service: Time/Date Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY (nuint16) NWPackTime (
    const NW_TIME N_FAR *sTime);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWPackTime
    (const sTime : pNW_TIME
) : nuint16;
```

Parameters

sTime

(IN) Points to the NW_TIME structure.

Return Values

Returns the packed time upon successful completion.

Remarks

The bits are defined as follows:

0-4	seconds divided by two
5-10	minutes
11-15	hours (0-23)

NWPackTime does no validity checking on the passed information. The programmer should ensure times in the associated structures are valid before passing them to NWPackTime.

See Also

[NWPackDate \(page 293\)](#), [NWPackDateTime \(page 295\)](#), [NWUnpackDate \(page 298\)](#), [NWUnpackDateTime \(page 300\)](#), [NWUnpackTime \(page 302\)](#)

NWUnpackDate

Unpacks a packed date into the passed structure

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98

Service: Time/Date Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY (void) NWUnpackDate (
    nuint16      date,
    NW_DATE N_FAR *sDate);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWUnpackDate
  (date : nuint16;
   Var sDate : NW_DATE
  );
```

Parameters

date

(IN) Specifies the date in packed format.

sDate

(OUT) Points to the NW_DATE structure.

Return Values

None

Remarks

Many functions return dates in a packed format identical to that defined by DOS. The bits are defined as follows:

0-4	Day
5-8	Month
9-15	Year minus 1980

See Also

[NWPackDate \(page 293\)](#), [NWPackDateTime \(page 295\)](#), [NWPackTime \(page 297\)](#),
[NWUnpackDateTime \(page 300\)](#), [NWUnpackTime \(page 302\)](#)

NWUnpackDateTime

Unpacks a packed date and time into the passed structures

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98

Service: Time/Date Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

void N_API NWUnpackDateTime (
    nuint32      dateTime,
    NW_DATE     N_FAR *sDate,
    NW_TIME     N_FAR *sTime);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWUnpackDateTime
  (dateTime : nuint32;
  Var sDate : NW_DATE;
  Var sTime : NW_TIME
  );
```

Parameters

dateTime

(IN) Specifies the date and time in packed format.

sDate

(OUT) Points to the NW_DATE structure (optional).

sTime

(OUT) Points to the NW_TIME structure (optional).

Remarks

Many functions return dates in a packed format identical to that defined by DOS. The time occupies the low order word and the date occupies the high order word. The bits are defined as follows:

0-4	Seconds divided by two
5-10	Minutes
11-15	Hours (0-23)
16-20	Day

21-24 Month
25-31 Year minus 1980

See [timedate.c](#) ([../../samplecode/clib_sample/misc_timedate/timedate.c.html](#)) for sample code.

See Also

[NWPackDate](#) (page 293), [NWPackDateTime](#) (page 295), [NWPackTime](#) (page 297),
[NWUnpackDate](#) (page 298), [NWUnpackTime](#) (page 302)

NWUnpackTime

Unpacks a packed time into the passed structure

NetWare Server: 3.11, 3.12, 3.2, 4.x, 5.x, 6.x

Platform: NLM, Windows NT, Windows 95, Windows 98

Service: Time/Date Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY (void) NWUnpackTime (
    nuint16      time,
    NW_TIME N_FAR *sTime);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWUnpackTime
    (time : nuint16;
     Var sTime : NW_TIME
    );
```

Parameters

time

(IN) Specifies the time in packed format.

sTime

(OUT) Points to the NW_TIME structure.

Remarks

The bits are defined as follows:

```
0-4      Seconds divided by two
5-10     Minutes
11-15    Hours (0-23)
```

See Also

[NWPackDate \(page 293\)](#), [NWPackDateTime \(page 295\)](#), [NWPackTime \(page 297\)](#),
[NWUnpackDate \(page 298\)](#), [NWUnpackDateTime \(page 300\)](#)

SecondsToTicks

Converts seconds to clock ticks

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Time/Date Manipulation

Syntax

```
#include <nwtime.h>

void SecondsToTicks (
    LONG    seconds,
    LONG    tenthsOfSeconds,
    LONG    *ticks);
```

Parameters

seconds

(IN) Specifies the number of seconds to convert.

tenthsOfSeconds

(IN) Specifies the tenths of seconds to convert.

ticks

(OUT) Points to the equivalent number of clock ticks.

Remarks

To convert clock ticks back to seconds, call the `TicksToSeconds` function.

One IBM* PC clock tick is approximately 1/18 second. (Eighteen [18.21] clock ticks equal approximately 1 second.)

See Also

[TicksToSeconds \(page 307\)](#)

strftime

Formats the time into an array under format control

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

size_t strftime (
    char          *s,
    size_t        maxsize,
    const char    *format,
    const struct tm *timeptr);
```

Parameters

s

(IN) Points to a character array.

maxsize

(IN) Specifies the maximum number of characters that can be placed in the array.

format

(IN) Points to the format control string.

timeptr

(IN) Points to the time argument.

Return Values

If the number of characters to be placed into the array is less than the value specified by the `maxsize` parameter, the number of characters placed into the array pointed to by the `s` parameter (not including the terminating NULL character) will be returned. Otherwise, a value of 0 is returned. If an error occurs, `errno` is set.

Remarks

`strftime` formats the time in the `timeptr` parameter into the array pointed to by the `s` parameter according to the `format` parameter and is not enabled for international use.

The `format` parameter string consists of zero or more directives and ordinary characters. A directive consists of a % character followed by a character that determines the substitution that is to take place. All ordinary characters are copied unchanged into the array. No more than the number of characters specified by the `maxsize` parameter are placed in the array.

Directive	Substitution
%a	Locale's abbreviated weekday name
%A	Locale's full weekday name
%b	Locale's abbreviated month name
%B	Locale's full month name
%c	Locale's appropriate date and time representation
%C	Century number ("19" if year is 1997)
%d	Day of the month as a decimal number (01-31)
%D	Date in the format mm/dd/yy (POSIX)
%h	Locale's abbreviated month name (POSIX)
%H	Hour (24-hour clock) as a decimal number (00-23)
%I	Hour (12-hour clock) as a decimal number (01-12)
%j	Day of the year as a decimal number (001-366)
%m	Month as a decimal number (01-12)
%M	Minute as a decimal number (00-59)
%n	Newline character (POSIX)
%p	Locale's equivalent of either AM or PM. If linked with NWPRE.OBJ, a %P generates AM and PM while the %p generates am and pm.
%r	12-hour clock time (01-12) using the AM/PM notation in the format hh:mm:ss (POSIX). If linked with NWPRE.OBJ, a %R generates AM and PM while the %r generates am and pm.
%S	Second as a decimal number (00-59)
%t	Tab character (POSIX)
%T	24-hour clock time in the format hh:mm:ss (POSIX)
%U	Week number of the year as a decimal number (00-52) where Sunday is the first day of the week
%w	Weekday as a decimal number (0-6) where 0 is Sunday
%W	Week number of the year as a decimal number (00-52) where Monday is the first day of the week
%x	Locale's appropriate date representation
%X	Locale's appropriate time representation
%y	Year without century as a decimal number (00-99)
%Y	Year with century as a decimal number
%Z	Timezone name, or by no characters if no timezone exists
%%	Character %

See Also

[asctime](#), [asctime_r](#) (page 272), [clock](#) (page 274), [ctime](#), [ctime_r](#) (page 279), [difftime](#) (page 281), [gmtime](#), [gmtime_r](#) (page 286), [localtime](#), [localtime_r](#) (page 288), [mktime](#) (page 291), [setlocale](#) (Internationalization), [time](#) (page 308)

Example

```
#include <stdio.h>
#include <time.h>

main()
{
    time_t time_of_day;
    char buffer[80];
    time_of_day = time (NULL);
    strftime (buffer, 80, "Today is %A %B %d, %Y",
              localtime (&time_of_day) );
    printf ("%s\n", buffer);
}
```

produces the following:

```
Today is Friday December 25, 1990
```

TicksToSeconds

Converts clock ticks to seconds

Local Servers: nonblocking

Remote Servers: N/A

Classification: 3.x, 4.x, 5.x, 6.x

Service: Time/Date Manipulation

Syntax

```
#include <nwtime.h>

void TicksToSeconds (
    LONG    Ticks,
    LONG    *seconds,
    LONG    *tenthsOfSeconds);
```

Parameters

ticks

(IN) Specifies the number of ticks to convert to seconds.

seconds

(OUT) Points to the number of seconds.

tenthsOfSeconds

(OUT) Points to the tenths of seconds.

Remarks

TicksToSeconds converts clock ticks to seconds and tenths of seconds. To convert seconds back to clock ticks, call the SecondsToTicks function.

One IBM PC clock tick is approximately 1/18 second. (Eighteen [18.21] clock ticks equal approximately 1 second.)

See Also

[SecondsToTicks \(page 303\)](#)

time

Returns the current calendar time

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

time_t time (
    time_t *timer);
```

Parameters

timer

(OUT) Points to the object into which the encoded calendar time (same as return value of `time`) has been placed.

Return Values

Returns the current calendar time encoded into type `time_t`. On error, returns -1 cast as type (`time_t`). To test for an error condition, compare the return value to (`time_t`) -1.

Remarks

The time represents the time in seconds since January 1, 1970 (Universal Coordinated Time). If the `timer` parameter is not NULL, the current calendar time is also stored in the object pointed to by the `timer` parameter.

See Also

[clock \(page 274\)](#)

tzset

Sets the `tzname`, `timezone`, and `daylight` global variable parameters

Local Servers: nonblocking

Remote Servers: N/A

Classification: Other

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

void tzset (void);
```

Return Values

None

Remarks

Before accessing the `tzname`, `timezone`, and `daylight` global variable parameters, you must first call `tzset` to initialize or update these variables to the correct values.

CLIB.NLM uses the time zone information that is set by issuing the SET TIMEZONE (also SET TIME ZONE) console command.

For NetWare 3.x, `tzset` always uses the time zone information that is active when CLIB.NLM loads. If the SET TIMEZONE command is issued after CLIB.NLM loads, `tzset` does not use the updated information. Therefore, for NetWare 3.x, the SET TIMEZONE command must be issued in the following prescribed order:

At the server console, issue the SET TIMEZONE command with the time zone name, hours, and daylight savings code parameters.

Load CLIB.NLM.

Load your NLM application.

Refer to the *Supervising the Network* manual for more information on the SET TIMEZONE command.

In NetWare 4.x, `tzset` uses the current time zone setting of the server to initialize or update the values of the `tzname`, `timezone`, and `daylight` global variable parameters. However, remember that these variables will not be automatically updated when the SET TIMEZONE command is issued. They are updated only when `tzset` is called. For this reason, you should call `tzset` before calling any of the time functions. Doing so ensures that the `tzname`, `timezone`, and `daylight` global variable parameters contain the proper values.

See Also

[asctime](#), [asctime_r](#) (page 272), [gmtime](#), [gmtime_r](#) (page 286), [localtime](#), [localtime_r](#) (page 288), [time](#) (page 308)

Example

```
#include <time.h>
#include <nwtime.h>
#include <stdio.h>

void print_zone()
{
    tzset();

    printf( "time zone names: %s %s\n", tzname[0], tzname[1]);
    printf( "timezone: %ld \n", timezone ) ;
    printf( "daylight: %d\n", daylight ) ;
}
```

produces the following:

```
time zone names: PST PDT
timezone: 28800
daylight: 1
```

Time/Date Manipulation Structures

13

This documentation alphabetically lists the time/date structures and describes their purpose, syntax, and fields.

NW_DATE

Contains the date in packed format

Service: Time/Date Manipulation

Defined In: nwmisc.h, nwmisc.inc, and nwncepxt.txt

Structure

```
typedef struct {
    nuint8    day ;
    nuint8    month ;
    nuint16   year ;
} NW_DATE;
```

Pascal Structure

```
NW_DATE = packed Record
    day : nuint8;
    month : nuint8;
    year : nuint16
End;
```

Fields

day

Specifies the day.

month

Specifies the month.

year

Specifies the year.

NW_TIME

Contains the time in packed format

Service: Time/Date Manipulation

Defined In: nwmisc.h, nwmisc.inc, and nwncepxt.txt

Structure

```
typedef struct {
    nuint8    seconds ;
    nuint8    minutes ;
    nuint16   hours ;
} NW_TIME;
```

Pascal Structure

```
NW_TIME = packed Record
    seconds : nuint8;
    minutes : nuint8;
    hours   : nuint16
End;
```

Fields

seconds

Specifies the seconds.

minutes

Specifies the minutes.

hours

Specifies the hours.

Remarks

Because the `hours` field is defined as a `nuint16`, `NW_TIME` takes up a `nuint32` of space.

tm

Contains time information

Service: Time/Date Manipulation

Defined In: time.h

Structure

```
struct tm {
    int    tm_sec ;
    int    tm_min ;
    int    tm_hour ;
    int    tm_mday ;
    int    tm_mon ;
    int    tm_year ;
    int    tm_wday ;
    int    tm_yday ;
    int    tm_isdst ;
};
```

Fields

tm_sec

Specifies the number of seconds after the minute. This number is in the range [0,61].

tm_min

Specifies the number of minutes after the hour. This number is in the range [0,59].

tm_hour

Specifies the number of hours after midnight. This number is in the range [0,23].

tm_mday

Specifies the day of the month. This number is in the range [1,31].

tm_mon

Specifies the months since January. This number is in the range [0,11].

tm_year

Specifies the number of years since 1900.

tm_wday

Specifies the number of days since Sunday. This number is in the range [0,6].

tm_yday

Specifies the number of days since January 1. This number is in the range [0,365].

tm_isdst

Specifies a Daylight Savings Time flag, defined as follows:

0	Daylight savings time is not in effect.
>0	Daylight savings time is in effect.
<0	Daylight savings time information is not available.

Remarks

The original values of the `tm_sec`, `tm_min`, `tm_hour`, `tm_mday`, and `tm_mon` fields are not restricted to ranges described for the `tm` structure. If these fields are not in their proper ranges, they are adjusted so that they are in the proper ranges. Values for the `tm_wday` and `tm_yday` fields are computed after all the other fields have been adjusted.

The value of the `tm_isdst` field depends upon whether you have linked your application to the `PRELUDE.OBJ` file or the `NWPRE.OBJ` file .

Some locales such Korean, Chinese, and Italian use the `tm_wday` structure field for the standard date format. If the `tm_wday` structure field is not set, an incorrect day will be displayed. An application which sets only the year, month, and day structure fields can compute the weekday by calling the C library function:

```
mktime (timePtr)
```


Variable Length Argument List Functions

14

This documentation alphabetically lists the variable length argument list functions and describes their purpose, syntax, parameters, and return values.

- [“va_arg” on page 318](#)
- [“va_end” on page 321](#)
- [“va_start” on page 322](#)

va_arg

Obtains the next argument in a list of variable arguments (macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Variable Length Argument List

Syntax

```
#include <stdarg.h>

type va_arg (
    va_list param,
    type);
```

Parameters

param

(IN) Specifies a variable argument.

type

(IN) Specifies the argument type.

Return Values

va_arg returns the value of the next variable argument, according to type passed as the second parameter.

Remarks

The macro va_arg can be used to obtain the next argument in a list of variable arguments. It must be used with the associated macros va_start and va_end. A sequence such as:

```
va_list curr_arg;
type next_arg;
next_arg = va_arg (curr_arg, type);
```

causes next_arg to be assigned the value of the next variable argument. The type is the type of the argument originally passed.

The macro va_start must be executed first in order to properly initialize the variable next_arg and the macro va_end should be executed after all arguments have been obtained.

The data item curr_arg is of type va_list which contains the information to permit successive acquisitions of the arguments.

See Also

[va_end](#) (page 321), [va_start](#) (page 322), [vfprintf](#), [vprintf](#) (Single and Intra-File Management), [vsprintf](#) (page 267)

Example

```
#include <stdarg.h>

void test_fn (const char *msg, const char *types, ...);

int main ()
{
    printf ("VA...TEST\n");
    test_fn ("PARAMETERS: 1, \"abc\", 546", "isi", 1, "abc", 546 );
    test_fn ("PARAMETERS: \"def\", 789", "si", "def", 789 );
}

static void test_fn (

const char *msg,      /* Message to be printed */
const char *types,   /* Parameter types (i,s) */
... )                /* Variable arguments    */

{
    va_list argument;
    int arg_int;
    char *arg_string;
    char *types_ptr;
    types_ptr = types;
    printf ("\n%s - %s\n", msg, types);
    va_start (argument, types);
    while (*types_ptr != '\0')
    {
        if (*types_ptr == `i`)
        {
            arg_int = va_arg (argument, int );
            printf ("integer: %d\n", arg_int );
        }
        else if (*types_ptr == `s`)
        {
            arg_string = va_arg ( argument, char *);
            printf ("string:  %s\n", arg_string);
        }
        ++types_ptr;
    }
    va_end (argument);
}
```

produces the following:

```
PARAMETERS: 1, "abc", 546
integer: 1
string: abc
```

```
integer: 546  
PARAMETERS: "def", 789  
string: def  
integer: 789
```

va_end

Completes the acquisition of arguments from a list of variable arguments (macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Variable Length Argument List

Syntax

```
#include <stdarg.h>
```

```
void va_end (
    va_list param);
```

Parameters

param

(IN) Specifies a variable argument.

Return Values

None

Remarks

The macro `va_end` is used to complete the acquisition of arguments from a list of variable arguments. It must be used with the associated macros `va_start` and `va_arg`. See the description for [va_arg \(page 318\)](#) for complete documentation on these macros.

See Also

[va_arg \(page 318\)](#), [va_start \(page 322\)](#), [vfprintf](#), [vprintf](#) (Single and Intra-File Management), [vsprintf \(page 267\)](#)

va_start

Starts the acquisition of arguments from a list of variable arguments (macro)

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

Service: Variable Length Argument List

Syntax

```
#include <stdarg.h>

void va_start (
    va_list param,
    previous);
```

Parameters

param

(IN) Specifies a variable argument.

previous

(IN) Specifies the previous argument being passed to the called function.

Return Values

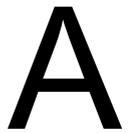
None

Remarks

The macro `va_start` is used to start the acquisition of arguments from a list of variable arguments. The `param` argument is used by the `va_arg` macro to locate the current acquired argument. The `previous` argument is the argument that immediately precedes the `"..."` notation in the original function definition. It must be used with the associated macros `va_arg` and `va_end`. See the description of [va_arg \(page 318\)](#) for complete documentation on these macros.

See Also

[va_arg \(page 318\)](#), [va_end \(page 321\)](#), [vfprintf](#), [vprintf](#) (Single and Intra-File Management), [vsprintf \(page 267\)](#)



Revision History

The following table outlines all the changes that have been made to the Program Management 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 strtok (page 257) and strxfrm (page 261) .
October 2002	Removed the strsub function (string.h). Updated the Pascal syntax for the structures.
May 2002	Updated the syntax of rand, rand_r (page 123) to remove the second parameter, which used to be the value parameter.
February 2002	Updated the Remarks section of set_app_data. Updated links.
September 2001	Added support for NetWare 6.x. Updated documentation for register_library. Changed the tzset (page 309) example to include <nwtime.h>.
June 2001	Changed reference to ExitThread to NXThreadExit in register_library. Made changes to improve document accessibility.
February 2001	Added documentation for ltime (page 290) , register_library, unregister_library,, strcasecmp (page 212) , strncasecmp (page 234) , strsep (page 250) , strsub, get_app_data, get_app_type, and set_app_data. Moved difftime (page 281) from Mathematical Computation Functions to Time/Date Manipulation Functions. Revised paragraph in each Time function with a _r variant, such as ctime and ctime_r to clarify the difference between the function operations and the purpose of the _r variant.
May 2000	Added revision history.
