NWLib 1.5

by Devont Software Inc.
18026 Deep Brook
Spring TX 77379
JimTyson@IX.NETCOM.COM
70421,1506 (Compuserve)
(713) 370-4215 (Fax)
(713) 370-0841 (24 Hour BBS)
www.wworks.com/~devont   (web page)

Table of Contents

New Functions

The following functions have been added to this relase:

NWLib

getMapInfo - collect information about a drive mapping.
getPathInfo - parse a path specification into distinct elements.

NWServer

createSemaphore - create/access a semaphore on the server.
freeSemaphore - release a server semaphore.
incSemaValue - increment a server semaphore value.
decSemaValue - decrement a server semaphore value.
querySemaphore - check a semaphore's station count/value.
serverLoginOK - check server login status.

NWPrint

getQueueJobList - create a formatted queue job listing.
getQueueJobNumbers - get a listing of queue job numbers.
getQueueJobInfo - get information about a specific queue job.
setQueueJobInfo - change queue job information.
setQueueJobPosition - move a queue job.
deleteQueueJob - delete a queue job.

Introduction

Welcome to NWLib - the first native Netware-aware VCL component for Borland's great new Delphi programming environment.

NWLib is a component which conveniently "wraps up" extremely complex Novell API calls and integrates them directly into the Delphi environment. The removal of the Netware layer allows the programmer to concentrate on more important levels of application design: the flow of data and the interface to the end user.

While NWLib publishes about 100 Netware specific functions, it actually contains over 300 Netware API calls, and dozens of internal structures. By digesting and combining the API calls and structures into more comprehensible pieces, NWLib makes it much easier to integrate full Netware functionality into your applications.

Sure, NWLib contains plenty of Novell Netware Bindery/NDS and connection information routines to make your network applications kick butt and take names. But even more, NWLib is the most compatible choice you can make in creating Netware functionality for your Delphi applications. Since NWLib makes calls directly to the client API kit installed on the client workstations and supplied by directly from Novell, you are assured maximum functionality and compatibility across all Netware platforms. Top that off with the fact that NWLib is pure optimized Delphi code, and you've got yourself a potent combination for speed, compatibility, ease of use and hard-to-beat functionality .

By simply dropping an icon or two onto a form, you instantly have access to familiar Netware commands such as WhoAmI, Map, Login, GetMemberList, SendLineMessage, GetDirRights, SList, Salvage, etc. You get four separate component units in which to include in your applications: File Server, Print Environment, Connection/Bindery/NDS and a general-purpose string manipulation toolkit. NWLib lets you concentrate what's really important : your data and the flow of your application.
So, what'll all this great convenience cost you? Amazingly, about a buck for each Netware function, and we throw in the general purpose toolkit free! Just think of how much time you'll save since you'll never have to crack the half-a-foot-thick Netware API manual set or try to decipher just what those guys up there in Utah burned into that goofy dynaText CD. NWLib sets you back about 80 bucks, and you can create as many applications as you want with it. One programmer, one license.

Believe me, the Novell API spec is no cake walk - it's something you definitely want to avoid if you can. It's awesomely rich, complex, and above all else, huge. I can't tell you how many late nights I've spent scratching my head only to find out a problem turned out to be "documentation omissions" or some other irritating item. When you think about it, you'd be crazy NOT to let someone else who specializes in Netware, and has a lot of time with the Novell API relieve you of this burden. For about the cost of an hour or two of your time, you get peace of mind and all your applications get a rich new set of tools.

What about updates? One of the good things about dealing with a small shop is that we don't mess around when it comes to fixes, updates and suggestions. You'll find a newer, registered version on our 24 hour BBS and/or WWW page that you can download free for 90 days or more. We only charge for an upgrade when a significant version changes with new features and capabilities that warrant a cost, and at such a price that you're willing and pleased to pay. You'll never be charged for bug fixes or patches that are certain to occur in any complex software product.

How about source code? NWLib's source code is purchasable seperately at a very modest cost. You must already own a copy of the compiled version of NWLib to purchase source code, of course.

So what if you use NWLib and don't send any money? The bad thing about that is that if you don't send in any money, you'll be using a version that displays an annoying little banner each time an instance of the library is created...but least your apps run without Delphi running....after all, you're writing network software that needs to be tested on multiple workstations, and Delphi is probably not installed on each of your network workstations. Your customers will see this banner if you distribute apps with NWLib demo code, and they'll think you're a cheapskate. No one wants to see that happen. You can't blame us for wanting to be paid for the enormous amount of time spent creating this software, you'd probably do the same if you wrote it yourself.

Above all else, enjoy yourself and have fun. If you're not having fun programming with Delphi, you're doing something wrong!

Jim Tyson - President
Devont Software Inc.
JimTyson@ix.netcom.com
24 Hour Fax: 713/370-4215
24 Hour BBS System: 713/370-0841
World Wide Web: www.wworks.com/~devont

Ordering NWLib

Please remember: one programmer, one license. If you've got a shop where many people can compile and code that contains our library code, you need to obtain a license for each possible instance. Additional copies of NWLib are only $25.00, so make sure to purchase an valid license for each programmer developing software that uses NWLib functions

If you do not provide a valid email address, we ship regular ground mail. Postage and handling charges are added to orders shipped ground mail.

United States:	05.00
Canada Mexico:	07.50
Europe:	10.00
Australia/NZ:	10.00
Asia:	12.00
Africa:	12.00

Our goal is to get the software in your hands as quickly as possible. If you send your order via electronic mail and your order does not arrive in your mailbox within two US working days, please contact us immediately. Sometimes our reply messages and attachments do not arrive to the correct address for a variety of Internet-related reasons. If we somehow recieved your message with an invalid address, we may not be able to reply back to you to tell you we failed in sending your files. If you don't hear back from us, something has happened to our messages.

If your Internet message fails to reach us, please try our Compuserve address over the internet: 70421,1506@compuserve.com Your messages are very valuable to us, and we think each deserves a speedy reply.

Payment Methods

1. American Express: Fax our order form to us directly: (713)370-4215. Please leave your complete American Express card and return email or shipping information. In most cases, you'll have your order waiting for you the next morning in your email box. You are charged $75.00 if we send via email.

2. CompuServe: go SWREG and use registration number 7641. Total in US dollars is $85.00. If you wish to order additional copies, you should contact us directly. In most cases, your order is waiting in your mailbox the next morning.

3. Personal/Company Checks:
   Devont Software Inc.
   18026 Deep Brook
   Spring TX 77379
   $75.00+shipping - US Currency on US Banks only.

4. Email: JimTyson@ix.netcom.com or 70421.1506@compuserve.com

Sorry, we do not accept MC/Visa. If you require alternative playment options, please contact us directly.

Functions by Category

TNWLib - Network Connection and Environment functions

TNWProp - Complete Object and Property management functions.

TNWServer - Server statistical, query and operating functions

TNWPrint - CAPTURE and related functionality

TNWNDS - Novell Directory Services name and context management

TNWTools - General Purpose utilities

Constants

```
const
  { Object Types }
  nw_user          = $0100 ;
  nw_group         = $0200 ;
  nw_printq        = $0300 ;
  nw_server        = $0400 ;
  nw_jobServer     = $0500 ;
  nw_gateway       = $0600 ;
  nw_printServer   = $0700 ;
  nw_archiveQueue  = $0800 ;
  nw_archiveServer = $0900 ;
  nw_jobQueue      = $0A00 ;
  nw_Administration= $0B00 ;
  nw_nasSNAServer  = $2100 ;
  nw_RemoteBridge  = $2600 ;
  nw_TCPIPGateway  = $2700 ;

  { Broadcast Modes }
  nw_caston        = 0   ;
  nw_castoff       = $01 ;
  nw_castserver    = $03 ;

  { Returned Path Formats }
  nw_format_netware       = 0 ;
  nw_format_server_volume = $01 ;
  nw_format_drive         = $02 ;
  nw_format_unc           = $03 ;

  { Internal Definitions }
  word_local              = 'Local'  ;

  { Bindery Types }
  type_set                = $02 ;
  type_item               = 0   ;

  { netware file open modes }
  nw_file_normal          = 0   ;
  nw_file_readOnly        = $01 ;
  nw_file_hidden          = $02 ;
  nw_file_system          = $04 ;
  nw_file_execute_only    = $08 ;
  nw_file_directory       = $10 ;
  nw_file_needsArchived   = $20 ;
  nw_file_shareable       = $80 ;

  { nds context keys }
  nds_key_flags           = $01 ;
  nds_key_confidence      = $02 ;
  nds_key_contextname     = $03 ;
  nds_key_transportType   = $04 ;
  nds_key_referralScope   = $05 ;

  nds_read                = 3  ;
  nds_compare             = 4  ;
  nds_search              = 6  ;
  nds_add_entry           = 7  ;
```

```
nds_modify_entry        = 9  ;
nds_read_attr_def       = 12 ;
nds_define_class        = 14 ;
nds_read_class_def      = 15 ;
nds_modify_class_def    = 16 ;
nds_search_filter       = 28 ;

nds_add_attribute       = $00 ;
nds_remove_attribute    = $01 ;
nds_add_value           = $02 ;
nds_remove_value        = $03 ;

nds_entry_browse        = $0001 ;
nds_entry_add           = $0002 ;
nds_entry_delete        = $0004 ;
nds_entry_rename        = $0008 ;
nds_entry_supervisor    = $0010 ;

nds_attr_compare        = $0001 ;
nds_attr_read           = $0002 ;
nds_attr_write          = $0004 ;
nds_attr_self           = $0008 ;
nds_attr_supervisor     = $0020 ;

nds_sms_scan            = $0001 ;
nds_sms_backup          = $0002 ;
nds_sms_restore         = $0004 ;
nds_sms_rename          = $0008 ;
nds_sms_delete          = $0010 ;
nds_sms_admin           = $0020 ;

max_rdn_chars           = 127 ;
max_dn_chars            = 254 ;
max_schema_name_chars   = 31  ;
max_rdn_bytes           = (2*(max_rdn_chars+1)) ;
max_dn_bytes            = (2*(max_dn_chars+1)) ;
max_schema_name_bytes   = (2*(max_schema_name_chars+1)) ;
max_asn1_name           = 31;
max_value               = (63 * 1024) ;
max_message             = $10000 ;
no_more_iterations      = -1 ; {0xffffffffl}

ftok_end                = 0  ;
ftok_or                 = 1  ;
ftok_and                = 2  ;
ftok_not                = 3  ;
ftok_lparen             = 4  ;
ftok_rparen             = 5  ;
ftok_aval               = 6  ;
ftok_eq                 = 7  ;
ftok_ge                 = 8  ;
ftok_le                 = 9  ;
ftok_approx             = 10 ;
ftok_aname              = 14 ;
ftok_present            = 15 ;
ftok_rdn                = 16 ;
```

```
ftok_basecls            = 17 ;

DSV_UNUSED_0                 = 0  ;
DSV_RESOLVE_NAME             = 1  ;
DSV_READ_ENTRY_INFO          = 2  ;
DSV_READ              = 3  ;
DSV_COMPARE                  = 4  ;
DSV_LIST              = 5  ;
DSV_SEARCH                   = 6  ;
DSV_ADD_ENTRY                = 7  ;
DSV_REMOVE_ENTRY             = 8  ;
DSV_MODIFY_ENTRY             = 9  ;
DSV_MODIFY_RDN        = 10 ;
DSV_DEFINE_ATTR              = 11 ;
DSV_READ_ATTR_DEF            = 12 ;
DSV_REMOVE_ATTR_DEF          = 13 ;
DSV_DEFINE_CLASS             = 14 ;
DSV_READ_CLASS_DEF           = 15 ;
DSV_MODIFY_CLASS_DEF         = 16 ;
DSV_REMOVE_CLASS_DEF         = 17 ;
DSV_LIST_CONTAINABLE_CLASSES     = 18 ;
DSV_GET_EFFECTIVE_RIGHTS   = 19 ;
DSV_ADD_PARTITION            = 20 ;
DSV_REMOVE_PARTITION         = 21 ;
DSV_LIST_PARTITIONS          = 22 ;
DSV_SPLIT_PARTITION          = 23 ;
DSV_JOIN_PARTITIONS          = 24 ;
DSV_ADD_REPLICA              = 25 ;
DSV_REMOVE_REPLICA           = 26 ;
DSV_OPEN_STREAM              = 27 ;
DSV_SEARCH_FILTER            = 28 ;
DSV_CREATE_SUBORDINATE_REF= 29 ;
DSV_LINK_REPLICA             = 30 ;
DSV_CHANGE_REPLICA_TYPE    = 31 ;
DSV_START_UPDATE_SCHEMA    = 32 ;
DSV_END_UPDATE_SCHEMA      = 33 ;
DSV_UPDATE_SCHEMA          = 34 ;
DSV_START_UPDATE_REPLICA   = 35 ;
DSV_END_UPDATE_REPLICA     = 36 ;
DSV_UPDATE_REPLICA         = 37 ;
DSV_SYNC_PARTITION         = 38 ;
DSV_SYNC_SCHEMA            = 39 ;
DSV_READ_SYNTAXES          = 40 ;
DSV_GET_REPLICA_ROOT_ID    = 41 ;
DSV_BEGIN_MOVE_ENTRY       = 42 ;
DSV_FINISH_MOVE_ENTRY            = 43 ;
DSV_RELEASE_MOVED_ENTRY    = 44 ;
DSV_BACKUP_ENTRY           = 45 ;
DSV_RESTORE_ENTRY          = 46 ;
DSV_SAVE_DIB               = 47 ;
DSV_UNUSED_2               = 48 ;
DSV_UNUSED_3               = 49 ;
DSV_CLOSE_ITERATION        = 50 ;
DSV_UNUSED_4               = 51 ;
DSV_AUDIT_SKULKING         = 52 ;
DSV_GET_SERVER_ADDRESS     = 53 ;
DSV_SET_KEYS               = 54 ;
```

```
DSV_CHANGE_PASSWORD        = 55 ;
DSV_VERIFY_PASSWORD        = 56 ;
DSV_BEGIN_LOGIN            = 57 ;
DSV_FINISH_LOGIN           = 58 ;
DSV_BEGIN_AUTHENTICATION   = 59 ;
DSV_FINISH_AUTHENTICATION  = 60 ;
DSV_LOGOUT                 = 61 ;
DSV_REPAIR_RING            = 62 ;
DSV_REPAIR_TIMESTAMPS            = 63 ;
DSV_CREATE_BACKLINK        = 64 ;
DSV_DELETE_EXTERNAL_REFERENCE    = 65 ;
DSV_RENAME_EXTERNAL_REFERENCE    = 66 ;
DSV_CREATE_ENTRY_DIR       = 67 ;
DSV_REMOVE_ENTRY_DIR       = 68 ;
DSV_DESIGNATE_NEW_MASTER   = 69 ;
DSV_CHANGE_TREE_NAME       = 70 ;
DSV_PARTITION_ENTRY_COUNT  = 71 ;
DSV_CHECK_LOGIN_RESTRICTIONS     = 72 ;
DSV_START_JOIN       = 73 ;
DSV_LOW_LEVEL_SPLIT        = 74 ;
DSV_LOW_LEVER_JOIN         = 75 ;
DSV_ABORT_LOW_LEVEL_JOIN   = 76 ;
DSV_GET_ALL_SERVERS        = 77 ;

{ Bindery Write Access Levels }
BS_ANY_WRITE    = 0 ;
BS_LOGGED_WRITE = $10 ;
BS_OBJECT_WRITE = $20 ;
BS_SUPER_WRITE  = $30 ;
BS_BINDERY_WRITE= $40 ;

{ Bindery Read Access Levels }
BS_ANY_READ     = 0 ;
BS_LOGGED_READ  = $01 ;
BS_OBJECT_READ  = $02 ;
BS_SUPER_READ   = $03 ;
BS_BINDERY_READ = $04 ;

{ Bindery Obj/Prop Flags }
BF_STATIC       = 0   ;
BF_DYNAMIC      = $01 ;
BF_ITEM         = 0   ;
BF_SET          = $02 ;

{ Trustee Rights Masks }
TA_NONE         = $00  ;
TA_READ         = $01  ;
TA_WRITE        = $02  ;
TA_OPEN         = $04  ;
TA_CREATE       = $08  ;
TA_DELETE       = $10  ;
TA_OWNERSHIP    = $20  ;
TA_SEARCH       = $40  ;
TA_MODIFY       = $80  ;
TA_ALL          = $A0  ;
```

```
{ Directory Inherited Rights Masks }
TR_NONE         = $0000 ;
TR_READ         = $0001 ;
TR_WRITE        = $0002 ;
TR_OPEN         = $0004 ;
TR_CREATE       = $0008 ;
TR_DELETE       = $0010 ;
TR_OWNER        = $0010 ;
TR_ACCESSCTRL   = $0020 ;
TR_FILESCAN     = $0040 ;
TR_MODIFY       = $0080 ;
TR_ALL          = $01FB ;
TR_SUPERVISOR   = $0100 ;
TR_NORMAL       = $00FB ;

{error mode constants (1.4+) }
NWERR_none               = $0000 ;
NWERR_addToStringList    = $0001 ;
NWERR_noMoreSearchDrives = $0002 ;
NWERR_newSearchDrive     = $0003 ;
{...}
NWERR_unknown            = $FFFF ;
```

Structures

TNWLib

TNWConnectInfo : NWLIB

General connection information interface structure. Passed by reference in function GetConnectInfo.

```
  type
    TNWConnectInfo = record
      serverConnID  : TNWConnHandle ;
      loginDateTime : TDateTime     ;
      internet      : string        ;
      sessionID     : word          ;
      ConnectID     : TNWConnNumber ;
      userID        : string        ;
      serverName    : string        ;
  end;
```

TNWPathInfo : NWLib

Holds all directory path information, as returned by parseNetwarePath.

```
 type
      TNWPathInfo = record
        nServer      : TNWConnHandle ;
        cServer      : string ;
        volumeID     : TNWVolNum ;
        volumeName   : string ;
        dirHandle    : TNWDirHandle ;
        pathOnly     : string ;
        relativePath : string ;
  end;
```

TNWDirRights : NWLib

Holds all effective rights for a directory (and files too for Netware 3.x and higher platforms).

```
    type
      TNWDirRights = record
        read          : boolean ;
        open          : boolean ; { Netware 2.x only }
        write         : boolean ;
        create        : boolean ;
        erase         : boolean ;
        modify        : boolean ;
        filescan      : boolean ; { Netware 3.x + only }
        accessControl : boolean ;
    end;
```

TNWParsedPath : NWLib

```
   type
     pTNWParsedPath = ^TNWParsedPath ;
     TNWParsedPath = record
       nServer      : TNWConnHandle ;
       serverName   : string[40]  ;
       volName      : string[128] ;
       dirPath      : string[128] ;
   end;
```

TNWMapInfo   :   NWLib

used by getMapInfo to obtain details about a drive mapping in effect on the workstation.

```
   type
    TNWMapInfo = record
       nServer     : TNWConnHandle ;
       serverName  : string[40]  ;
       fullPath    : string[128] ;
       driveStatus : word  ;
   end;
```

TNWPrint

TNWQueueJobInfo   : TNWPrint

Holds information about print queue jobs. Used in getQueueJobInfo.

```
    type
      TNWQueueJobInfo = record
        nServer         : TNWConnHandle ;
        cQueue          : string        ;
        jobID           : TNWQueueJobID ;
        ownerName       : string        ;
        serverName      : string        ;
        queueServerName : string        ;
        jobFileName     : string        ;
        jobDescription  : string        ;
        workstationID   : longint       ;
        entryDateTime   : TDateTime     ;
        execDateTime    : TDateTime     ;
        jobPosition     : word          ;
        jobFlags        : TNWQueueJobCtrlFlags ;
    end;
```

TNWQueueJobCtrlFlags   : TNWPrint

embedded in TNWQueueJobInfo. Allows you to change certain print queue job status flags, such as Hold.

```
    type TNWQueueJobCtrlFlags = record
      auto_start      : boolean ;
      entry_restart   : boolean ;
      entry_open      : boolean ;
      user_hold       : boolean ;
      operator_hold   : boolean ;
    end;
```

TNWServer

TNWFileInfo   : NWServer

General File System structure. When displaying information about a Netware-based file with GetFileInfo, you'll pass along this strucure by reference so Netware can fill it up for you.

```
    type
      TNWFileInfo = record
        name              : string        ;
        updatedBy         : string        ;
        ownerID           : string        ;
        lastArchivedBy    : string        ;
        creationdate      : TDateTime     ;
        lastArchiveDate   : TDateTime     ;
        lastAccessDate    : TDateTime     ;
        lastModified      : TDateTime     ;
        updateDateTime    : TDateTime     ;
        fileSize          : longint       ;
        inheritedRights   : TNWRightsMask ;
        maximumSpace      : TNWDirSpace   ;
        attributes        : TNWAttributes ;
        flags             : TNWFlags      ;
        nameSpace         : TNWNameSpace  ;
        nameLength        : TNWNameLen    ;
    end;
```

TNWConnStats   : NWServer

GetUserStats uses this structure to obtain connection-status information about any particular user on the network.

```
    type
      TNWConnStats = record
        loginTime      : TDateTime     ;
        bytesRead      : TNWNumBytes   ;
        bytesWritten   : TNWNumBytes   ;
        totalRequests  : TNWNumPackets ;
        recordLocks    : TNWNumPackets ;
        fileLocks      : TNWNumPackets ;
        expirationTime : TDateTime   ;
    end;
```

TNWServerInfo : TNWServer

GetServerStats uses this structure to provide detailed information about the file server and it's operating environment.

Note that some of the values are valid in a Netware 4.x environment only. The Novell Client SDK does not include functions to obtain this information on any other platform.

```
    type
      TNWServerInfo = record
        serverName        : string ;
        serverUpTime      : TNWSysTime  ;    {4.x only}
        processor         : byte ;
        numprocs          : byte ;
        utilization       : byte ;           {4.x only}
        PacketsIn         : TNWNumPackets ;  {4.x only}
        packetsOut        : TNWNumPackets ;  {4.x only}
        version           : string ;
        maxConns          : TNWNumber ;
        ConnsInUse        : TNWNumber ;
        maxConnsUsed      : TNWNumber ;
        numVolumes        : TNWNumber ;
        sftLevel          : TNWSupportLevel ;
        ttsLevel          : TNWSupportLevel ;
    end;
```

TNWVolInfo : TNWServer

VolInfo fills this stucture with information regarding a particular server volume. The Novell Client SDK provides this functionality only on a Netware 2.2 platform only.

```
    type
      TNWVolumeInfo = record
        sysUpTime         : TNWSysTime  ;
        volumeNumber      : TNWVolNum   ;
        logicalDriveNum   : TNWDriveNum ;
        sectorsPerBlock   : TNWNumber   ;
        startingBlock     : TNWNumber   ;
        totalBlocks       : TNWNumber   ;
        availableBlocks   : TNWNumber   ;
        totalDirSlots     : TNWNumber   ;
        availableDirSlots : TNWNumber   ;
        maxDirSlotsUsed   : TNWNumber   ;
        isHashing         : TNWFlags    ;
        isCaching         : TNWFlags    ;
        isRemovable       : TNWFlags    ;
        isMounted         : TNWFlags    ;
        volName           : TNWVolName  ;
    end;
```

TNWDiskCacheInfo   :   TNWServer

Use GetCacheInfo along with this structure to obtain detailed information about your server's disk caching environemtn.   The Novell Client SDK provides this functionality only on Netware 2.2 platforms.

```
    type
      TNWDiskCacheInfo = record
        serverUpTime              : TNWSysTime ;
        cacheBufferCount          : TNWNumber ;
        cacheBufferSize           : TNWNumber ;
        dirtyCacheBuffers         : TNWNumber ;
        cacheReadRequests         : TNWNum ;
        cacheWriteRequests        : TNWNum ;
        cacheHits                 : TNWNum ;
        cacheMisses               : TNWNum ;
        diskReadRequests          : TNWNum ;
        diskWriteRequests         : TNWNum ;
        diskReadErrors            : TNWNumber ;
        diskWriteErrors           : TNWNumber ;
        cacheGetRequests          : TNWNum    ;
        cacheFullWriteRequests    : TNWNum    ;
        cachePartialWriteRequests : TNWNum    ;
        backgroundDirtyWrites     : TNWNum    ;
        backgroundAgedWrites      : TNWNum    ;
        totalCacheWrites          : TNWNum    ;
        cacheAllocations          : TNWNum    ;
        thrashingCount            : TNWNumber ;
        LRUDirtyBlockCount        : TNWNumber ;
        readBeyondWriteCount      : TNWNumber ;
        fragmentedWriteCount      : TNWNumber ;
        cacheHitOnUnavailableCount: TNWNumber ;
        cacheBlockScrappedCount   : TNWNumber ;
    end;
```

TNWFileSysInfo : TNWServer

GetFileSysStats uses this structure to return information about the server's file i/o activity. The Novell Client SDK provides this functionality only on Netware 2.2 platforms.

```
    type
      TNWFileSysInfo = record
        serverUpTime              : TNWSysTime ;
        maxOpenFiles              : TNWNumber  ;
        maxFilesOpened            : TNWNumber  ;
        currOpenFiles             : TNWNumber  ;
        totalFilesOpened          : TNWNum     ;
        totalReadRequests         : TNWNum     ;
        totalWriteRequests        : TNWNum     ;
        currChangedFatSectors     : TNWNumber  ;
        totalChangedFatSectors    : TNWNum     ;
        fatWriteErrors            : TNWNumber  ;
        fatalFatWriteErrors       : TNWNumber  ;
        fatScanErrors             : TNWNumber  ;
        maxIndexFilesOpen         : TNWNumber  ;
        currOpenIndexedFiles      : TNWNumber  ;
        attachedIndexFiles        : TNWNumber  ;
        availableIndexFiles       : TNWNumber  ;
    end;
```

TNWMemCacheInfo : TNWServer

GetCacheInfo fills this structure with information regarding a file server's caching activities in a Netware 4.x environment. GetCacheInfo has no effect on any other Netware release.

```
    type TNWMemCacheInfo = record
       serverUpTime              : TNWSysTime ;
       writeBlockCount           : longint ;
       diskWriteCount            : longint ;
       writeErrorCount           : longint ;
       numCacheHits              : longint ;
       numDirtyCacheHits         : longint ;
       cacheDirtyWaitTime        : longint ;
       cacheMaxConcurrentWrites  : longint ;
       maxDirtyTime              : longint ;
       DirCacheBuffers           : longint ;
       maxByteCount              : longint ;
       minCacheBuffers           : longint ;
    end;
```

TNWDeletedFileInfo   :   TNWServer

GetFileInfo uses this structure to obtain information on files stored on Netware volumes.

```
    type
      TNWDeletedFileInfo = record
        attributes       : TNWAttributes ;
        flags            : TNWFlags      ;
        nameSpace        : TNWNameSpace  ;
        nameLength       : TNWNameLen    ;
        name             : string        ;
        creationdate     : TDateTime     ;
        ownerID          : string        ;
        lastArchiveDate  : TDateTime     ;
        lastArchivedBy   : string        ;
        updateDateTime   : TDateTime     ;
        updatedBy        : string        ;
        fileSize         : longint       ;
        inheritedRights  : TNWRightsMask ;
        lastAccessDate   : TDateTime     ;
        deletedDateTime  : TDateTime     ;
        deletedBy        : string        ;
    end;
```

TNWProp

TNWRights: General File and Directory Rights Structure   :   TNWProp

```
type
  TNWRights   =   record
        supervisor       : boolean ;
        read             : boolean ;
        open             : boolean ; {Netware 2.x Only}
        write            : boolean ;
        create           : boolean ;
        erase            : boolean ;
        modify           : boolean ;
        filescan         : boolean ; { Search in Netware 2.x}
end;
```

TNWNDS

TNWDSAttrRights   :   TNWNDS

```
type
  TNWDSAttrRights = record
        compare          : boolean ;
        read      : boolean ;
        write     : boolean ;
        self      : boolean ;
        supervisor : boolean ;
end;

type
  TNWDSEntryRights = record
        browse  : boolean ;
        add     : boolean ;
        delete  : boolean ;
        rename  : boolean ;
        supervisor: boolean ;
end;

type
  TNWDSSMSRights = record
        scan    : boolean ;
        backup  : boolean ;
        restore : boolean ;
        rename  : boolean ;
        delete  : boolean ;
        admin   : boolean ;
end;
```

Source Code

NWLib source code is availalble to any registered NWLib Library user. You can purchase the full set of the NWLib code, including all Pascal routines and Netware API encapsulation routines and structures for only $100.00. This license enables one concurrent programmer to view/modifiy the source code or use the NWLib source code text files in a project. This license does not impose on the licensing rights of the current NWLib library license already in effect at the site.

Additional NWLib Source Code licenses are only $40.00. You must order additional source code licenses directly from us.

You can purchase NWLib source code via Compuserve's SWREG Service Number 9731. Or, send us your American Express card information to our fax number at (713)370-4215, or internet your information to JimTyson@IX.NETCOM.COM.

We also accept signed, numbered and faxed company purchase orders from our registered United States and Canadian users, or those with major headquarters located in the US. We'll send an invoice due in 15 days.

Index

#
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

Glossary

B

Bindery Object Types

addObjectToSet : Boolean

Netware Objects can contain properties that are of the BF_SET type.   This property type can contain many entries, unlike the BF_ITEM property type, which can contain only a single entry.

Use this function to add a new entry into an object's   property which is defined as a BF_SET type.

Parameters

nServer: TNWConnHandle   The server connection handle that owns the object to modify.   Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objName : String.   The name of the object that contains the BF_SET property to modify.

propName: String.   The name of the BF_SET property in which to modify.   This property must already exist for the objName.

memberName: String.   The new entry to add into the set-type property.   The property is added to the end of the list.

Returns

Boolean.   True if the member is added to the set-type property.   False usually means the server/objName combination is not correct, or the property name does exist for the specified user.

Example

```
if addObjectToSet(      0,
                 'BARNEY',
                 'KEYS',
                 'F-0930938384') then
  okBox('Keys Checked Out to Barney Successfully') ;
```

See Also

isMember, addUserToGroup, modifyTrusteeRights, getPropertyList

addUserToGroup : boolean

You can use this function if you need to add a new user into an existing Netware group on a server. A user must posess a Bindery Write Access Level of BS_SUPER_WRITE to the group AND user objects in order to add the user to the group.

When a user is added to a group, two item property sets are actually modified: The user's login name is added to the group's 'GROUP_MEMBERS' property, and the group name is added to the user's 'GROUPS_I"M_IN' property.

You could actually perform these exact same steps with NWLib's own AddObjectToSet function call, but it's much easier to make a single function call which takes care of this in one simple step.

Parameters

nServer: TNWConnHandle The server connection handle that owns the user and group objects in which to modify. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

groupName : String The name of the group in which a new member is to be added.

userName: String The name of the user which is to be added to the specified group.

Returns

Boolean. True if the both user and group properties are modified and the user is added to the group.

Example

```
if addUserToGroup(     0,
                  'EVERYONE',
                  'BILLY') then
     okBox('Message from Server:    ' + getServerName(0) +
        ';Billy Added to the group Everyone!') ;
```

See Also

deleteUserFromGroup
deleteObjectFromSet
createObject

Capture : boolean

In a Netware environment, as you probably already know, you use the CAPTURE utility to redirect printing to shared network devices. NWLib's Capture function performs the same function as the Netware command-line utility, except you can call this function internally within your own applications.

There are a few differences in how to call the API function vs. the command-line utility that Netware provides: whereas the Netware CAPTURE utility requires that you place the capture settings on the command line, such as "/nb /nff" etc., NWLib requires that you first create the capture environment, then apply the properties to the environment.

In the example below, the following events occur to create a valid capture environment:

1. A TNWCaptureFlagss record structure and integer type variable are declared.
2. The CaptureFlags record structure is filled with current Capture environment settings if they exist. Otherwise, it is filled with default capture data.
3. Changes are made to the CaptureFlags environment.
4. The Capture statement is executed.
5. The CaptureFlags environment was applied to the capture environment.
6. End of function.

Parameters

nServer : TNWConnHandle. The server in which you want to handle print jobs.

cQueueName : String. The name of the Netware Print Queue you want to redirect.

nPort : TNWLpt. The port number. IPX/NETX networks can handle LPT1 to LPT3. A VLM Environment can handle up to LPT9.

Returns

Boolean. True if the capture was successful.

Example

```
var
  captureFlags : TNWCaptureFlags ;
  nLPT : TNWLpt ;
  oldqueue : string ;
begin
    nLPT := 1 ;
    if getCaptureFlags(nLPT,captureFlags) then
       begin
         captureFlags.formfeeds := False ;     { /nff }
         captureFlags.banner   := '' ;                { /nb }
         captureFlags.tabSize := 0 ;                { /nt   }
         oldqueue := captureFlags.qname ;     { i.e. Save old Capture, if you need }
         Capture('fs1','printer1',nLPT) ) then
           begin
                setCaptureFlags(GetPrimaryServerID,nLPT,captureFlags) ;
                okBox('Print Redirected to FS1/Printer1') ;
           end;
```

```
          end;
end ;
```

Hint

Note there are two read-only elements in the TNWCaptureFlags record structure. If a Capture environment is already in place and you fill the TNWCaptureFlags structure using NWLib's getCaptureFlags function, you can read two additional items: the name of the print queue currently in use, and the server name to which it belongs, in addition to all the other print attributes currently in use.

You can use these elements to restore print environments to their original state once you have completed whatever task you need in your programs.   Simply go back to the record structure, read the elements, then perform another Capture/setCaptureFlags call, and viola, back we go.   The old queue name and server are stored until you call another getCaptureFlags function call using the same TNWCaptureFlags record structure.

See Also

endCap, setCaptureFlags, getCaptureFlags, isCaptured, getBannerUserName, setBannerUserName, getMaxPrinters

changeNWPassword : boolean

Use this function to change a user object's login password. You must be a supervisor or equivalent to use this function, or you must know the user's old login password in order to change it.

No confirmation is performed, so use this function with care!

Parameters

nServer : TNWConnHandle : The server connection handle that owns the object who's password is to be changed. Pass a 0 (zero) as the server connection handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : String. The name of the user that gets the password change.

cPassword: String. The new password. Not case sensitive. Note the password must comply with the user's account restrictions for size and uniqueness.

cOldPassword: string. The user's Old password. If you have Supervisor-level access to this object, you can supply an empty string as this parameter.

Returns

Boolean. True if the operation is successful. False usually means the user object specified is not valid on the specified server, or the user does not have sufficient object rights in order to call this function.

Example

```
if changeNWPassword(       getPrimaryServerID,
                          'guest',
                          'bigRedSwitch',
                                                          'think') then
   okbox('Password Changed!') ;
```

See Also

getPrimaryServerID
getServerHandle

changeObjectSecurity : boolean

Use this function to promote or demote a user's Bindery Read/Write access level. A user with Supervisor Read/Write object security can change anyone's password, full name, or any other property data.

A common use for this function is granting a regular user higher bindery rights to modify or add other objects, without neccesarily granting SUPERVISOR equivilency, or promoting rights to a particular user so they can perform modifications on other object's properties, without granting extra access privileges or equivilency that may be undesirable.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object in which the Bindery Read/Write access level is to be changed. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objName: String Pass the name of the user object in which the bindery access level is to be changed.

readSecurity: TNWFlags Specifies the Bindery Read Access level to grant to the object. The NWLib include file contains a complete listing of available Bindery Read Security constants, such as BS_ANY_READ and BS_LOGGED_READ.

writeSecurity: TNWFlags Specifies the Bindery Write Access level to grant to the object. The NWLib include file contains a complete listing of available Bindery Write Security constants, such as BS_ANY_WRITE and BS_SUPER_WRITE.

Returns

Boolean. True if the object's Bindery Access Levels were changed. False usually indicates the objectname does not exist on the specified server, or the user making the call does not have SUPERVISOR access rights on the desired server.

Example

```
if changeObjectSecurity(     getServerHandle('FS2'),
                         'GUEST',
                         BS_LOGGED_READ,
                         BS_ANY_WRITE) then
   okBox('FS2/GUEST Bindery Access Levels Changed!') ;
```

{now guest can read data about any object on the server, and only write to other objects whose Write access level is set as BS_ANY_WRITE, which means just about no one.}

See Also

changePropertySecurity, getObjectInfo

changePropertySecurity : boolean

Use this function when you want to promote or demote the Bindery Access Read/Write levels that an object must posess in order to view or change the information contained in the property.

An object's default properties also contain a standard Bindery Access Read/Write Security level, such as a user's 'IDENTIFICATION' property, which is of BS_LOGGED_READ and BS_OBJECT_WRITE level.   This means that any logged in object can read the user's full name, but only the object or anyone with at least BS_OBJECT_WRITE access rights to the object can edit the user's full Name.   A SUPERVISOR has BS_SUPER_READ and BS_SUPER_WRITE, which is the highest obtainable Bindery Access Levels, except for the file server itself.

Parameter

nServer: TNWConnHandle   The server connection handle that owns the object in which new property read/write access levels are to be edited.   Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

Returns

True if the propertie's access levels are modified.   False usually indicates a bad server/userName combination, or insufficient object read/write access levels.

Example

```
if changePropertySecurity(      0,
                                'GUEST',
                                'IDENTIFICATION',
                                BS_LOGGED_READ,
                                BS_SUPER_WRITE) then
   okBox('Now only a SUPERVISOR can change GUEST's FullName!') ;
```

See Also

deleteProperty
createProperty
changeObjectSecurity
getPropertyList

createObject : boolean

With this function, you can create new Netware users, groups, print servers, or any other valid Netware object types (defined in the NWLib include file).

Standard Netware properties are automatically created for each valid object type. In other words, if you are creating a regular Netware user account, the standard properties such as 'IDENTIFICATION', 'GROUPS_I'M_IN', 'LOGIN_CONTROL', and 'SECURITY_EQUALS'. This emulates the functionality of SYSCON, so objects created by this function can be used in a normal environment with no further property modifications or additions. The same is true for Group Objects, Print Queue and Server Objects, etc.

However, you should note that for all new objects, default properties created are not actually populated with the default property information you expect when you create the object using standard Netware tools, such as SYSCON or NetAdmin. Therefore, if you want a user to be included in the EVERYONE group, you'll simply need to make a call to the addUserToGroup function after the user object is successfully created. The same is true for similar user defaults, such a mailbox directory { usually SYS:\MAIL\getObjID() } and sufficient access rights to it, and a home directory. NWLib gives you the power to do all these things easily yourself in minutes, without requiring any other library or toolkit.

Parameters

nServer: The server connection handle that will own the new object. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objectName : String The name of the new object to create. Not case sensitive.

objType : TObjType A valid Netware object type, as defined in the NWLib include file (nw_user, nw_group, etc.).

fullName: String The object's new Full Name. Some object types do not contain a full name property, in which case this parameter will be ignored. Pass an empty string if you do not wish to specify the object's full name.

password: String The object's new login password, if applicable. Pass an empty string if you want to prompt the user on initial login to the server.

permanent: Boolean If you want to create an object that exists until you physically delete it, specify true. Otherwise, the object is automatically removed from the file server when the server is brought down. A normal user created with the standard Netware tools is a permanent user that exists until removed.

readSecurity: TNWFlags The object's read access level. A user reading information about the user must contain a read Access level equal or greater than the user's read access level in order to obtain the information. The standard user read access level is BS_LOGGED_READ, meaning only logged-in users can read public information about this object. The complete listing of object read access levels is specified the NWLib include file.

writeSecurity: TNWFlags Just as an object must contain a read access level, it must also contain a write access level. A user must have a write access level equal to or greater than the object's write access level to modify the object in any way. The standard user read access level is BS_SUPER_WRITE, which means only a SUPERVISOR can change or delete the object. The complete listing of object write access levels is specified the NWLib include file.

Returns

Boolean. True if the new object is created on the specified server. False usually indicates an invalid server connection handle or insufficient rights on the specified server.

Example

```
if createObject(          0,
                'newUser1',
                nw_user,
                'New User Number One',
                'zippitydodahday',
                true,
                BS_LOGGED_READ,
                BS_SUPER_WRITE)   then
   okBox('User Created Successfully!') ;
```

See Also

deleteObject
addUserToGroup
writeItemProperty
changeObjectSecurity
changePropertySecurity
modifyTrusteeRights

createProperty : boolean

All Netware objects contain at least one property automatically associated with it when the object is created by standard Netware tools. For instance, a standard Netware User Object contains several, such as 'IDENTIFICATION', which holds the user's full name, and 'GROUPS_I''M_IN'. Each standard Netware object type contains a unique set of default properties which are usually created at the time the object is created.

Properties can be either BF_ITEM or BF_SET types. BF_ITEM properties contain a SINGLE ENTRY up to 128 characters in length. For instance, a user's IDENTIFICATION property is of type BF_ITEM, since you can create only a single full name for the user. Conversely, a user property such as 'GROUPS_I'M_IN' is of the BF_SET Property type, since a User Object can belong to more than one Netware group.

You can create any object property you desire. Netware ignores any properties that do not relate to the operation of the network itself.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object in which the new property is to be created. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objectName: String The name of the object that receives the new property.

propertyName: String. If creating a Netware property, ensure it exactly matches the required property as specified in the Novell API documentation, or any third-party books on Netware internals. If creating a new property that is ignored by Netware, specify any continuous string, without spaces or ascii characters greater than 128. Not case sensitive, as it will be converted to uppercase.

propertyTypeFlag: TNWFlags BF_ITEM or BF_SET

permanent: Boolean Specify true if the property is to remain on the server until it is physically deleted. False means the property is automatically deleted when the network server is brought down.

readSecurity: TNWFlags Specifies a bindery access level that a user must posess in order to read the specified property. A complete listing of the read security constants is contained in the NWLib include file. A standard for most Netware properties is BS_LOGGED_READ, which means any logged-in user can read the information contained in the specified property. Of course, the user must also have sufficient rights to the object itself in order to read the object in the first place.

writeSecurity: TNWFlags Specifies a bindery access level that a user must posess in order to write information into the specified property. For example, a user's 'IDENTIFICATION' property is created with the BS_OBJECT_WRITE access level, meaning the user or supervisor object can change the user's full name property. The object's write access level takes precedence over the property write access level, which means an object must have the required access rights in order to write to the object itself, no matter what the property access level specifies.

Returns

Boolean. True if the property is created successfully for the specified object.

Example

```
if createProperty       (0,
              'GUEST',
              nw_user,
              'ENTRY_DOORS_ALLOWED',
              true,
              BS_OBJECT_READ,
              BS_SUPER_WRITE) then
  OKBox('User Property Successfully Created!') ;
```

See Also

Paragraph

createSemaphore : TNWSemaHandle ;

Creates or accesses a 'flag' on the server, with a specified number of available slots.   You can use semaphores for such utility functions as limiting the number of simultaneous accesses to a network resource or program.   The file server handles the details of the semaphore, such as freeing the slot after a workstation disconnects from the network, etc.

If the semaphore already exists, the station count using the semaphore is incremented.   If the maximum number of stations is already using the semaphore,   zero is returned.

A Netware semaphore resource also contains a 'value' data value which you can assign any way you like, using the incSemaValue and decSemaValue functions.   Initially, Netware assigns the max. station count parameter to this value.

Parameters

nServer : TNWConnHandle.   The server in which to store the semaphore.

SemaphoreName :   string.   The name of the semaphore to create/use on the server.

MaxInstances: word.   The maximum number of stations which can access this semphore at the same time.   For instance, if you specify '13' as this parameter, no more than 13 workstations can open this semaphore at the same time.   This number is also used for the beginning semaphore 'value', which can be used to store additional flag information of your choice.

The MaxInstances number is ignored unless it is the inital call to the named semaphore on the server.

Returns

TNWSemaHandle.   The handle to the new semaphore.   This handle must be used to free the semaphore resource when the module ends.   If no more stations slots are available on the specified semaphore, a 0 (zero) returns.

Example

```
var
  semaHandle :   TNWSemaHandle ;
begin
   semaHandle := createSemaphore(0,'NWLIB',10) ;
   if (semaHandle > 0) then
       begin
           winExec('notepad.exe g:\secret.txt') ;
           freeSemaphore(0,semaHandle) ;
       end
   else
       alertBox('Max Stations Reached!') ;
end;
```

See Also

freeSemaphore
querySemaphre

incSemaValue
decSemaValue

decSemaValue : boolean

Decreases the arbitrary value of the specified semaphore.

Parameters

nServer: The server connection handle which owns the semaphore to decrement.

semaHandle: TNWSemaHandle. The semaphore handle to decrement.

amount: The amount to subtract from the semaphore value.

Returns

Boolean. True if the call is successful and the semaphore value is decremented the specified amount.

Example

```
if decSemaValue(0,semaHandle,5) then
      close ;
```

See Also

incSemaValue

deleteObject : boolean

To complete remove an existing Netware object, you simply make one function call, and NWLib takes care of the rest.   On success, the specified object is completely eradicated from the file server.

Parameters
nServer: TNWConnHandle   The server connection handle that owns the object   to be deleted Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objName: String.   The name of the object in which to delete.

Returns
Boolean.   True if the object is deleted.   False usually indicates the user making the call is not a SUPERVISOR, or the object does not exist on the specified server.

Example
```
if deleteObject('BUZZLITE') then
     okBox('Woody Was Here') ;
```

See Also
renameObject, createObject

deleteObjectFromSet : boolean

In order to remove an entry from an object's property that is defined as a BF_SET type, you'll need to use this function call. As you probably already know, an object's property can be defined as either BF_ITEM or BF_SET. BF_SET property types can contain many individual entries, such as a group object can contain many members in the 'GROUP_MEMBER' set-type property
.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object in which the entry is be remove from the set-type property. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objName: String Specify the object containing the BF_SET type property that contains an entry that needs to be deleted.

propName: String Specify the property name that contains the entry you'd like to remove.

memberName: String. The actual entry that you'd like to remove from the BF_SET property.

Returns

Boolean. True if the member is removed from the set-type property. False usually indicates a bad server/objectName combination, or you do not have a sufficient bindery write access level to the property to delete the entry.

Example

```
if deleteObjectFromSet(      0,
                        'Laser_1',
                        'Q_OPERATORS',
                        'JOEY') then
   okBox('Joey Is No Longer a Queue Operator;;' +
         'That Should Teach Him to Delete our Print Jobs!!') ;
```

See Also

addObjectToSet
deleteUserFromGroup
deleteProperty
deleteTrusteeRight

deleteProperty : boolean

Use this function to complete remove any type of property that a Netware object contains.

You should not delete an object's default property types, such as a user's 'IDENTIFICATION' or 'GROUPS_I''M_IN' properties. Doing so will cause error messages to appear in Netware tools such as SYSCON, or may cause other programs to fail to recognize valid objects if they expect particular properties to be present.

Parameters
nServer: TNWConnHandle The server connection handle that owns the object in which the property is to be deleted. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objName: String. The name of the object that contains the property to be deleted.

propName: String. The name of the property to delete.

Returns
Boolean. True on success or False if something fails, such as inadequate bindery access levels for the object and/or property.

Example
```
if deleteProperty(        getServerHandle('FS1'),
                  'DEBRA',
                  'keys') then
   okBox('Key Property Removed from User Debra') ;
```

See Also
deleteObjectFromSet
writeItemProperty
createProperty
getPropertyList

deleteQueueJob : boolean

Removes a job from a print queue.

Parameters

queueJobInfo : TNWQueueJobInfo.   A structure containing at least the server connection handle, the print queue name, and the job ID.

You must supply at least the first three elements of the queueJobInfo structure in order to properly call this function.

Returns

Boolean.   True if the prinb job was deleted from the queue.   False usually indicates one or more of the first three elements of queueJobInfo are invalid.

Example

```
var
   tempList : TStringList ;
   queueJobInfo : TNWQueueJobInfo ;
   serverConn : TNWConnHandle ;
   queueName : string ;
begin
      serverConn := getPrimaryServerID ;
      queueName := 'laser1' ;
      templist := TStringList.create ;
        if getQueueJobList(serverConn,queueName,tempList) then
            begin
                  queueJobInfo.nServer := serverConn ;
                  queueJobInfo.cQueue := queueName ;
                  queueJobInfo.jobID := TNWQueueJobID(tempList.objects[0]) ;
                  if deleteQueueJob(queueJobInfo) then
                        okBox('Queue Job Deleted!') ;
            end;
    tempList.free ;
end;
```

See Also

getQueueJobInfo
setQueueJobInfo
getQueueJobList
getQueueJobNumbers

deleteTrusteeRight : boolean

Allows a user with SUPERVISOR rights to delete a Netware object's trustee access right. These rights control the amount of access privileges an object posesses to a particular volume/directory.

Parameters

nServer : TNWConnHandle. The Server connection handle that owns the object in which the trustee rights are to be deleted.

Pass a 0 (zero) to this function and NWLib will automatically use the 'primary' server connection handle (same as getPrimaryServerID returns).

cUserName: String. Specify the Login Name of the user to modify.

cVolume: String. Specify the name of the volume on the server that contains the directory. You can specify a blank parameter ('') if the complete volume:directory path is included in the cPath parameter.

cPath: String. Specify the complete directory path. You can include a volume in the path string if the cVolume parameter is empty.

Returns

Boolean. True if the operation was successful on the specified user. False if an error occurs. Likely problem is insufficient bindery access rights to the specified object.

Example

```
if deleteTrusteeRight(  getPrimaryServerID,
                        'GUEST',
                        '',
                        'sys:public') then

    okBox('Trustee Right Deleted Properly');
```

See Also

getServerHandle
getPrimaryServerID
modifyTrusteeRights
getTrusteeList

deleteUserFromGroup : boolean

Use this simple and quick function call to remove a user from a specified group on any server attached to the workstation.

You must have BS_SUPER_WRITE or access levels to the user and group objects in order to perform this function.

Parameters

nServer: TNWConnHandle The server connection handle that owns the user and groups to modify. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

groupName: String. The name of the group containing the user to delete.

userName: String. The name of the user to remove from the group.

Returns

Boolean. True if the user is removed from the desired group. The user then loses all connection with the group, including trustee rights and all other priviledges assigned to the group object.

Example

```
if deleteUserFromGroup(     getPrimaryServerID,
                            'EVERYONE',
                            'arnoldziffle') then
     okBox('User Removed from the Group') ;
```

See Also

addUserToGroup
deleteObject
deleteObjectFromSet

disableLogins : boolean

Use DisableLogins when you need to be sure no one logs into a file server. For instance, you may be performing a VREPAIR or something, and want to make sure that no one gets lost in lotus-land while this is occurring.

Users which are currently logged in are not affected by this function.

Be careful! if you disable logins, logout and then leave the network with no active Console Operators in which to re-enable logins, you'll have no recourse but to use the physical file server console in which to type in the 'Enable Logins' command. That might sound easy, but what if we're dealing with a server accessible only over a T1 connection 2500 miles away? "Uh, yes...this is Jim from Houston. Do you have anyone over there that can enable logins to the file server for me? I accidentally locked everyone out. Yeah. Uh-huh. OK, press Alt Esc.................. [tickitytickitytypinginthebackground].....Try Holding down the alt key. It says A-L-T on it..."

Parameters

nServer: TNWConnHandle. The server connection handle in which to disable logins. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

Returns

Boolean. True if the server logins were actually disabled. If users try to attach to the server, the Netware shell returns an error message and refuses attachment to the server.

Example

```
if disableLogins(getPrimaryServerID) then
   OKBox('Server Logins Disabled!') ;
```

See Also

enableLogins, NWLogout, NWLogin

DownServer : Boolean

DownServer immediately brings down a Netware File Server without any warnings or confirmations. Any open files are immediately closed by the Netware operating system, but any unsaved changes still at the workstation will not be saved.

You must be a Console Operator to bring a file server down using this function.

Parameters

nServer: TNWConnHandle ;

Returns

Boolean. True if the file server was actually brought down.

Example

```
var
   cserver: string ;
begin
   cserver :=   getServerName(getPrimaryServerID)) ;
   if YesNoBox('Bringing Down Server ' + cserver) and
        downServer(getPrimaryServerID) then
              okBox('File Server ' + cserver + ' Downed!) ;
end;
```

Sub Heading

Paragraph

enableLogins : boolean

Perform this command to enable logins to the file server, if they had previously been disabled via the NWLib disableLogin function, or through the standard netware console command.

You must be a console operator in order to enable logins to a file server.

Parameters

nServer : TNWConnHandle.   The server connection handle in which to enable logins.

Returns

Boolean.

Example

```
if enableLogins(getPrimaryServerID) then
   okBox('Server Logins Enabled!') ;
```

See Also

disableLogins, NWLogin

endCap : boolean

Use the endCap function when you want to terminate a Capture environment already in place on the workstation. The specified port is returned to non-redirected mode, so if any print data arrives at the port, it will be directed to the local device.

Parameters

nPort : TNWLpt. The LPT port number.

Returns

Boolean. True if the endCap was successful. False means the port is not valid on the workstation.

Example

```
if endCap then
   okBox('Local Print Mode Restored') ;
```

See Also

Capture, setCaptureFlags, getCaptureFlags

freeSemaphore : boolean

Releases the resources associated with a semaphore on the file server and decrements the active station count on the semaphore, allowing other workstations to access the semaphore.

You should always free a semaphore before allowing it to fall off scope.   Otherwise, the semaphore flag will not be cleared until the workstation clears the connection to the server.

If the workstation is the last using the semaphore, the file server automatically clears any allocation left for the semaphore.

Parameters

nServer: TNWConnHandle.   The server connection handle that owns the semaphore.
Pass a 0 (zero) as this parameter to default to the current server.

semaHandle: TNWSemaHandle.   The Semaphore handle to clear.

Returns

Boolean.   True if the semaphore is cleared and the station count is decemented.
False indicates a bad semaphore or server handle.

Example

```
if freeSemaphore(semaHandle) then
   close ;
```

See Also

createSemaphore

fullName : string

Use this function to query the Netware Bindery or NDS system (at the current context) and return the full name of any user logged in at any connected server.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to query Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : string. The name of the user you want to query.

Returns

String: The full name of the specified user on the current server. The full name can be specified in Netware using the SysCon or NetAdmin programs.

Example

```
edit1.text := FullName(0,WhoAmI(0)) ;
```

See also

NDSWhoAmi

getBannerUserName : string

Use getBannerUserName to return a string containing the workstation's current Capture environment Banner User Name.

Parameters

none.

Returns

String. The current Banner User Name.

Example

```
var
   cbanner : string ;
begin
   cbanner := getBAnnerUserName ;
   okBox('Name Printed on Banners: ' + cBanner) ;
end;
```

See Also

setBannerUserName, Capture, setCaptureFlags

getBinderyList : TStringList

Create a listbox or StringGrid, then call this function once and instantly fill up your list with all network users, the names of the print queues on the current server, group names, etc. The return value is a TStringList, so you can use the output directly, without any fuss or re-entrant calls to the Netware API. NWLib takes care of all the messy details for you.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

nObjType : TObjType. The type of object to return. Use nw_user, nw_group, nw_printq, etc. See the NWLIB include file for a complete listing of all possible netware object types.

Returns

A TStringList full of the objects you want. All neatly lined up and ready to be placed into an object that contains a TStringList type property, or acted upon iterively in your programs.

Example

```
var
  myList : TStringList;
  ntemp : word ;
begin
  myList := TStringList.Create ;
  mylist := GetBinderyList(GetPrimaryServerID,nw_user);
  for ntemp := 1 to myList.Count do begin
     okBox(mylist[ntemp-1] + ' is a User on this Network') ;
  end;
  winSuper.userList.items.addStrings(mylist) ;
end;
```

getBroadcastMode : boolean

getBroadcastMode tells you if the workstation has disabled network broadcast notification messages using the CASTOFF command or other means (such as NWLib's own setBroadcastMode function).

Each server stores it's own setting for a user's preference for broadcast mode.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

Returns

Boolean: True if the workstation can receive network broadcasts, and False if they have been disabled.

Example

```
if getBroadcastMode then
   caston.checked := True ;
```

getCacheInfo : boolean   (Netware 4.x)

When you need to check out a server's memory cache statistics, create a timer object, then have it grab them by making a call to getCacheInfo every 1 or 2 seconds.   A rich set of statistics is returned in your referenced structure.

This function can only be used on connections to Netware servers 4.x and higher.

Parameters

nServer : TNWConnHandle
    The server connection handle to query.   Must be a 4.x server.

var cacheInfo: TNWMemCacheInfo
    Your referenced record structure.   If the call is successful, it is filled with cache statistics.

Returns

Boolean.   True if the call is successful and the referenced data structure is populated with information.

Example

```
var
   memCacheInfo : TNWMemCacheInfo ;
begin
   if getCacheInfo(getPrimaryServerID,memCacheInfo) then
      okBox('Server Write Count: ' + intToStr(memCacheInfo.diskWriteCount)) ;
end;
```

See Also

getDiskCacheStats, getServerStats, getUserStats

getCaptureFlags : boolean

Use getCaptureFlags to initialize a TNWCaptureFlags record structure with current capture environment settings, such as the print queue name, number of copies, etc. You can use this record structure to pass to a new Capture environment to set that Capture environment up just like the old one, or make changes to the structure, then pass it along to another Capture environment to adjust the settings any way you desire.

Parameters

nPort : TNWLpt. The local LPT port number. Usually 1 to 3.

VAR CaptureFlags : TNWCaptureFlags. The record structure, passed by reference, which will contain the various capture environment settings.

Returns

Boolean. True if the record structure was properly filled. Note that if a capture environment is not in place on the specified port, default information is placed into the record structure, but the queue name will be empty and the server connection handle value will be 0. You can test for these conditions, or use the isCaptured function to test for a non-captured port.

Example

```
var
  captureFlags : TNWCaptureFlags ;
  nLPT : TNWLpt ;
  oldqueue : string ;
begin
    nLPT := 1 ;
    if getCaptureFlags(nLPT,captureFlags) then
       begin
         captureFlags.formfeeds := False ;    { /nff }
         captureFlags.banner   := '' ;               { /nb }
         captureFlags.tabSize := 0 ;               { /nt   }
         oldqueue := captureFlags.qname ;    { i.e. Save old Capture, if you need }
         Capture('fs1','printer1',nLPT) ) then
           begin
                setCaptureFlags(GetPrimaryServerID,nLPT,captureFlags) ;
                okBox('Print Redirected to FS1/Printer1') ;
           end;
       end;
end;
```

Sub Heading

Paragraph

getConnectedServerList : TStringList

When you need to get a listing of servers in which the current workstation is currently logged in, drop in this function.   You'll get back a TStringList in which you can use as the foundation of a visual control, or loop through and do custom processing on each element.

Parameters

None

Returns

TStringList: A complete listing of servers in which the current workstation is logged into.

Example

```
var
   templist : TStringlist ;
   nloop : byte
begin
   templist := TStringList.Create ;
   tempList.AddStrings(getConnectedServerList) ;
   for nloop := to templist.Count do begin
         okBox('Server Name: ' + templist[nloop-1]) ;
   end;
end;
```

getConnectID : TNWConnHandle

getConnectID returns a workstation's connection number to the default file server. This can range from 1 to the maximum number of Novell workstation connection licenses you own.   For example, a 250 user Netware system's workstation connection numbers can range from 1 to 250.   You need to pass a workstation connection number when attempting to obtain information about the workstation.

Parameters

nServer: TNWConnHandle.   The connection handle of the server in which to query.   Pass a 0 (zero) as this parameter and the default server connection handle is used.

Returns

nConnNumber : TNWConnHandle.   The workstation connection number to the default file server.

Example

```
myConnNumber := GetConnectID(0) ;
```

getConnectInfo : boolean ;

Use this function to obtain general-purpose information about an active connection on a file server.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

nConn: TNWConnNumber. The user's connection number to the server you want to query.

VAR connectInfo : TNWConnectInfo. A referenced structure that contains general purpose information about the desired connection.

Returns

Boolean. True if the referenced structure is filled with general purpose connection information.

Example

```
var
   connectinfo : TNWConnectinfo ;
   nConns: word ;
   connList : TConnList ;
   nloop : word ;
begin
   getUserConnList(getPrimaryServerID,'JIM',nconns,connList) ;
   for nloop := 1 to nconns do begin
        if getConnectInfo(getPrimaryServerID,connList[nloop-1],connectinfo) then
             okBox('Found Net Address: ' + connectInfo.internet) ;
   end;
end;
```

See Also

getUserConnList

getDeletedFileInfo : boolean ;

You'll use getDeletedFileInfo when you want to return information about a file that has been deleted from a network volume. Important information, such as the file deletor, size, and delete date/time is obtained from a referenced structure by making this single function call.

Of course, the calling workstation must have sufficient rights to see any directory's deleted file information. If not, False is returned and the referenced structure is not filled with information.

Parameters

cfile: string
The complete path and filename of the deleted entry.
The file must have been deleted from a network volume.

deletedFileInfo :TNWDeletedFileInfo
A predefined structure passed by reference. On successful return, this structure is passed back to you filled with all kinds of interesting information about the file.

Returns

Boolean. True if the call found the file and obtained information.

Example

```
var
  deletedFileInfo : TNWDeletedFileInfo ;
begin
  if getDeletedFileInfo('z:\help.nfo',deletedFileInfo) then
     okbox('File: ' + deletedFileInfo.name) ;
end;
```

See Also

Paragraph

getDeletedFiles : boolean ;

You send getDeletedFiles a server connection handle and a path specification, and you get back a TStringList of all deleted files that are still recovereable. Very handy if you are writing a Salvage interface utility, for instance.

Parameters

nServer : TNWConnHandle. The server that contains the deleted file list.
cPath: string. The complete path and file specification. Wildcards OK.

Returns

TStringList: The result of the search using the path and file specification. Since this return value is a TStringList, you can easily incorporate the information into your listBoxes and stringGrids.

Example

```
begin
  listbox1.addStrings(getDeletedFiles(getPrimaryServerID,'z:\*.*')) ;
end;
```

See Also

getDeletedFileInfo, getVolFileList, getFileInfo, getVolumes

getDiskCacheStats ; boolean

If you are connected to a Netware 2.2 server, you can call this function to obtain detailed information about the server's disk caching activities.

Parameters

nServer : TNWConnHandle.   The connection handle of the Netware 2.2 server.

var diskCacheInfo : TNWDiskCacheInfo.   A referenced structure that holds the statistical information if the call is successfully performed.

Returns

Boolean.   True if the referenced array is filled with disk caching information.

Example

```
var
   diskCacheInfo : TNWDiskCacheInfo ;
begin
   if getDiskCacheStats(getPrimaryServerID, diskCacheInfo) then
        okBox('Disk Writes: ' + intToStr(diskCacheInfo.diskWriteRequests)) ;
end;
```

See Also

getServerStats, getFileSysStats, getVolinfo

getEffectiveRights : boolean

When obtaining an object's access level to a directory, Netware uses several factors to compute the true access level granted, such as the Inherited Rights Mask and/or volume restrictions. Although specific access rights may be granted to the object, these rights may be decreased depending on such factors.

NWLib provides this function so you know precisely the current object's computed access rights to a particular directory when placed in the context of the environment at hand. Good information to have when your program needs to know if the user can write to a particular directory or not.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to query. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

pathName: String The complete pathname, including the volume or drive letter in which the effective rights are to be computed.

VAR rightsList: TNWRights A predefined record structure containing each of the available rights in the directory.

Returns

Boolean. True if the call is successful and the referenced variable is populated with computed effective rights to the specified directory.

Example

```
var
  rightsList : TNWRightsList ;
begin
  result := false;
   getEffectiveRights(0,
              'sys:/apps/einstein/workfiles',   {or g:\apps\....} }
              rightsList) ;
  if (rightsList.write or
     rightsList.supervisor)   then
        result := true ;
end;
```

See Also

getObjectDirRights
modifyTrusteeRights
deleteTrusteeRight

getFileInfo : boolean ;

getFileInfo can give you details about a file that exists on a Netware volume. Some file information, such as the file's Owner and archiver Object Name, can only be obtained by making a call to this function.

Parameters

cfile : string

The complete pathname to the desired file. Wildcard are NOT allowed.

fileInfo : TNWFileInfo

A structure passed by reference which contains the details about the file. If the call is successful, you can obtain data from the fields of this structure.

Returns

boolean. True if the call is successful and elements are written into the referenced data structure.

Example

```
var
  fileInfo : TNWFileinfo
begin
  if getFileInfo('z:\public\net$log.dat',fileInfo) then
     okbox('I got: ' + fileInfo.name) ;
end;
```

See Also

getDeletedFileInfo, getVolFileList, getVolumes

getFileSysStats : boolean

If you are connected to a Netware 2.2 server, you can use this function to obtain detailed information about a file server's file system activities.

Parameters

nServer : TNWConnHandle.   The server to query.

var fileSysInfo : TNWFileSysInfo.   A referenced array that holds all statistical information if the call is successful.

Returns

Boolean.   True if the call is successful and the referenced array is filled with the server's file system statistical information.

Example

```
var
  fileSysInfo : TNWFileSysInfo ;
begin
  if getFileSysStats(getPrimaryServerID, fileSysInfo) then
     okbox('Server Read Requests: ' +
                    intToStr(fileSysInfo.totalReadRequests)) ;
end;
```

See Also

getServerStats, getUserStats

getFirstNetDrive : char

getFirstNetDrive simply returns the first Netware drive letter encountered. You can use this to determine where to place a user after a logout command is executed.

Parameters

None.

Returns

Char: the first drive letter encountered. This can vary, depending on the workstation's LASTDRIVE setting (config.sys) and parameters specified in the NET.CFG file.

Example

```
var
   chtemp : char ;
begin
   chtemp := getFirstNetDrive;
   if map(chtemp,'fs1/sys:') then
      OKBox('Good Job!') ;
end;
```

getMapInfo : boolean

Retrieves information about a drive mapping, such as the host server, path and drive status flags.

Parameters

drive : char. The drive letter to query.
var mapInfo : TNWMapInfo. Filled with infomation relating to the mapped resource upon successful execution of the function.

Returns

Boolean. True on success. False usually indicates the drive is not a shared resource.

Example

Paragraph

See Also

map
mapShow
mapDelete

getMaxPrinters : byte

getMaxPrinters simply returns the maximum number of printer ports supported by the current version of the Netware shell. IPX/NETX networks support LPT1 to LPT3, and most VLM environments support ports up to LPT9.

Parameters

none.

Returns

Byte. The maximum number of ports available on the workstation.

Example

```
var
   nports : byte
begin
   nports := getMaxPrinters ;
   if (nports > 0) then
         okBox('You have ports!;;I'd have that looked at if I were you.') ;
end;
```

See Also

Capture, setCaptureFlags, getCaptureFlags, isCaptured

getMemberList : TStringList

In one simple function call, you can get the complete contents of a Novell Group into a TStringList. You can use this list as a source for a TListbox or TStringGrid, or you can create a loop and process each of the items in the list.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cGroupName: string. The name of the group you want to list. EVERYONE is a common group name in Netware.

Returns

TStringList. A listing of the group. One member on each line of the TStringList.

Example

```
var
  myList : TStringList ;
  ntemp : word ;
begin
   mylist := TStringList.Create;
   myList := GetMemberList(GetPrimaryServerID, 'EVERYONE') ;
   for ntemp := 1 to myList.count do begin
      okbox(myList[ntemp-1] + ' is an EVERYONE Member') ;
   end;
end;
```

getMyGroups :   TStringList ;

Returns a listing of groups in which a user is a member.   You must have console operator rights to return the names of groups a user belongs to other than yourself.

Since the return value of this function is in TStringList format, it's very easy to incorporate its output into onscreen objects such as Listboxes, StringGrids, etc.

Parameters

nServer: TNWConnHandle   The server connection handle that owns the object   to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : string.   The user who's groups to obtain.   You must be a console operator if this cUserID is not you.

Returns

TStringList. The complete contents of the user's group membership list.

Example

```
begin
  listbox1.addStrings(getMyGroups(GetPrimaryServerID,'JIM')) ;
end;
```

See Also

isConsoleOperator, isMember

getNextNetDrive : char

getNextNetDrive returns the next free non-local drive letter. You can use this to obtain a drive letter which is not already mapped.

Parameters

None.

Returns

char: The drive letter of the first non-local network drive. Any local devices are skipped.

Example

```
var
  chtemp : char ;
begin
  chtemp := GetNextNetDrive;
  if Map(chTemp,'fs1/sys:') then
      okBox('Way to Go!') ;
end;
```

getObjectDirRights : boolean

Use getObjectDirRights when you need to know exactly which directory rights are granted to a Netware object.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to query. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objectName: String The object name whose directory rights are to be obtained.

pathName: String The complete pathname for which the object's trustee rights are obtained. You must include either the volume or drive letter in the fully-qualified path.

VAR rightsList: TNWRights A predefined record structure containing the object's native trustee rights at the given directory.

Returns

Boolean. True if the call is successful, and the referenced TNWRights structure is filled with directory rights to the given pathname.

Example

```
var
  rightsList : TNWRightsList
begin
getObjectDirRights(0,
               'STEVEG',
               'sys:public',     {or z:\public, or z:.\, etc...}
               rightsList) ;
if rightsList.supervisor then
  okbox('STEVEG has SUPERVISOR access to SYS:PUBLIC!') ;
```

See Also

getEffectiveRights
deleteTrusteeRight
modifyTrusteeRights

getObjectInfo : boolean

Use this function to determine an objects read/write security level, and if the object contains any defined properties.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to query. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objectName: String. The name of the object to query.

VAR hasProperties: Boolean. On successful execution of the function, this variable contains a boolean value. True if the specified objectName actually contains valid properties, or False if no properties exist for this object.

VAR readSecurity: TNWFlags. On successful execution of the function, this variable contains objectName's readSecurity access level. User, Group and PrintQueue objects usually contain a BS_LOGGED_READ read access security level, which means any user logged into the server can read information from default properties.

VAR writeSecurity: TNWFlags. On successful execution of the function, this variable contains objectName's writeSecurity access level. User, Group and PrintQueue objects usually contain a BS_OBJECT_WRITE write access security level, which means only the objectName itself or a SUPERVISOR can make changes to the default properties. An object's IDENTIFICATION property is BS_OBJECT_WRITE by default, which allows users to change their own Full Name property.

Returns

Boolean. True if the function obtains the information properly from the Netware server. False usually indicates the nServer/objectName parameters are not correct in the specified scope.

Example

```
var
  hasProps:  Boolean ;
  readAccess,
  writeAccess : TNWFlags ;
begin
  if getObjectInfo(      0,
                 'SUPERVISOR',
                 hasProps,
                 readAccess,
                 writeAccess) then
  begin
     case readAccess of
       BS_ANY_READ :
                 okbox('Anyone Can Read') ;
       BS_LOGGED_READ :
                 okbox('Only Logged-In Users Can Read') ;
       BS_OBJECT_READ :
                 okBox('Only the Object and Supervisors Can Read') ;
```

```
      BS_SUPER_READ :
              okbox('Only Supervisors Can Read') ;
    end;
  end;
end;
```

Sub Heading

Paragraph

getObjectNumber : TObjID

getObjNumber is a basic building block function which the native Netware Object ID number, as it is stored in the Bindery/NDS. For the most part, you'll probably use this function only in conjunction with GetObjName, which requires the output of this function as a parameter. NWLib calls this function frequently to obtain data handles to specific objects in the Bindery/NDS.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objectName : string. The objectName you want to query. Like SUPERVISOR, or PRINTER1.

nObjType : TObjType. nw_user, nw_group, nw_printq, etc. A Complete listing of possible object types can be found in NWLib's include file.

Returns

TObjID: a long integer, as it is stored in the Netware Bindery/NDS.

Example

```
ntemp := NWGetObjNumber(0,'SUPERVISOR',nw_user) ;
if (ntemp > 0) then
   okBox('Supervisor is a valid Netware User') ;
```

getObjID : string

Returns an object's gives corresponding hexidecimal Object ID obtained from the Bindery or NDS (if the object exists within the current context)..

Netware automatically creates a directory for each user in the SYS:MAIL directory. The name of those directories directly corresponds to the numbers that you receive back from this function.

It's important to note that Object ID's are not syncronized between servers, except the SUPERVISOR user in the Netware 3.x world always has an Object ID of '1' ... for whatever that's worth.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

UserName : string The User's Login Name which to query.

objType : TObjType The object type of the given UserName, such as nw_user, nw_group, etc. NWLib's include file contains each of the possible object types.

Returns

String: The user's Object ID, represented in hexidecial string format. Common examples of the returned value are; 600001, 5000001, B7200046. etc. The number is guaranteed unique to each user on a single server.

Example

```
var
  cfile : string
begin
  cfile := 'f:\mail\' + GetObjID(getPrimaryServerID,
                          'NEIL',nw_user) + '\2112.PEW' ;
  if fileExists(cfile) then
    okBox('TaDa!') ;
end;
```

See Also

NDSGetObjID

getObjName : string

Returns the Object's Name from a user's Raw Netware Object ID.

Unless you are reading Netware objects or properties directly, you'll probably never need to use this function. NWLib automatically calls this function internally to convert data from the Bindery/NDS into a more human-readable format.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to query Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

nObjID : TObjID. This is a longint number, in lo-hi bit order. This is the native format Novell uses to store Object information.

Returns

String: a UserID, like SUPERVISOR, JIMT or DEBRA.

Example

```
var
   ntemp : longInt ;
   ctemp : string ;
begin
   ntemp := 1029384;
   ctemp := getObjName(0,ntemp) ;   {'Billy' or something, if valid }
end;
```

getObjType : TObjType

Use this function to determine a Netware object's data type, such as a User, Group, Print Queue or File Server.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be deleted Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : string The UserID you need to query, such as SUPERVISOR.

Returns

The type of the user, such as nw_user, nw_group, nw_printQ or nw_server, etc. Full object type constants are located in the NWLib include file.

Example

```
if getObjType(0,'SUPERVISOR') = nw_user then
   okBox('This Network Has a Guy Named SUPERVISOR on it') ;
```

See Also

getObjID
NDSGetObjID

getPathInfo: boolean

Separates individual path items from a string. Allows for any valid path specification, such as:

server/vol:path
drive:path
vol:path
path
file

Parameters

pathSpec: string. The complete path specification, in any valid directory path format.

Returns

boolean. True if the path is parsed properly. Note that this function does not verify if the path actually exists.

Example

```
var
   pathInfo : TNWPathInfo ;
begin
      if getPathInfo('z:\public',pathInfo) then
            okBox('Server Name: ' + pathInfo.serverName) ;
end;
```

See Also

parseNetwarePath

getPrimaryServerID : TNWConnHandle

GetPrimaryServerID returns the workstation's connection handle to the default network server. A lot of Netware function calls are 'server centric' and can return different information from server to server. So, when you call these functions, you need to pass the server connection handle so NWLib knows which server you intend to query for the results. Use this function when you simply want the action to be performed on the default calling server, and do not want to call a server specifically.

On login, the server in which the login executes is considered the 'default' or 'preferred server' connection. You can specify a preferred server in the boot-up NET.CFG or SHELL.CFG file. See your network documentation for more information on these configuration files.

Parameters

None

Returns

TNWConnHandle: The connection handle to the default server. While there's not much you can do with a server connection handle, there are plenty of other NWLib functions that can.

Example

```
myServerHandle := GetPrimaryServerID;
```

getPropertyList : TStringList

To obtain a listing of an object's properties, you can make one call to this function, and you'll get back a TStringList containing the name of each of the properties which have been created for a specific object on the network.

You can use the output of this getPropertyList to fill listboxes and grids, or iterate through the list yourself to perform special processing, without the need to create linked lists or pointers.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to query. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

SearchValue: String. You can pass a single wildcard (asterisk: *) character to obtain ALL properties for a particular object, or you can pass letters and a wildcard (gr*) to obtain a listing of properties that begin with a particular series of characters. Or, you can pass an entire string of characters to obtain the propery name that matches your input, such as 'IDENTIFICATION'.

Returns

TStringList: A completely initialized TStringList object that you can use as the basis for listboxes, string grids, etc.

Example

```
{assume you have a TListBox named PROPS on your form}

props.items.addStrings(
                getPropertyList( getPrimaryServerID,
                                 'SUPERVISOR',
                                 '*')
                ) ;


                --- or ---

if (getPropertyList(     0,
                'SUPERVISOR',
                'identification').count < 1) then
     alertBox('SUPERVISOR's INDENTIFCATION Property is Gone!') ;
```

See Also

getTrusteeList
deleteProperty
createProperty
writeItemProperty

getQueueJobInfo : boolean

Retrieves detailed information about a print queue job.

Parameters

jobInfo : TNWQueueJobInfo. A structure containing the server name, queuename, and queue job ID. The remainder of the fields are filled in upon successful completiong of the function.

Returns

Boolean. True if the queue job ID was read properly. False usually indicates the queue job ID, server or queue name is invalid. When reading live print queues, it's possible the queue job ID was removed from the queue right before your call the getQueueJobInfo, or right after the information is read.

Example

```
var
  queueJobInfo:    TNWQueueJobInfo ;
   ntemp : byte ;
   serverConn : TNWConnHandle ;
   queueName : string ;
begin
    serverConn := getPrimaryServerID ;
    queueName := 'laser' ;
    if getQueueJobList(serverConn,queueName,,jobList) then
        begin
            queueJobInfo.nServer := serverConn ;
            queueJobInfo.cQueue := queueName ;
            ntemp := 0 ;
            while (jobList[ntemp] >0) do begin
                 queueJobInfo. jobID := jobList[ntemp] ;
                 if getQueueJobInfo(jobInfo) then
                       okBox('Job Owner: ' + queueJobInfo.ownerName) ;
                inc(ntemp) ;
            end;
       end;
end;
```

See Also

getQueueJobNumbers
getQueueJobList
setQueueJobInfo

getQueueJobList : boolean

Creates a fullly-formatted StringList displaying the contents of a specified print queue on a server. You get the same information you'd expect t see in PrintCon or any other Netware print queue query program.

Parameters

nServer: TNWConnHandle. The server which owns the queue in which to query. You can pass a 0 (zero) as this parameter to indicate the current server.

cQueue: String. The name of the print queue to query.

var JobList: TStringList. An initialized TStringList object. It is filled with print queue job information obtained from the specified queue. Any existing elements in the StringList are removed prior to filling the list with new values. Each element in JobList also contains the corresponding job's Job ID number, which you can use in other NWLib calls, such as getQueueJobInfo. Typecase each element's 'object' to a TNWQueueJobID to extract it from the list. See the example below for detail.

Returns

Boolean. If the StringList is modified, True is returned. False usually indicates an incorrect server handle or queue name. True is returned and an empty stringList is returned if the print queue happens to be empty.

Note this function returns a 'snapshot in time.' That is, the print queue jobs that exist at the time this call is performed are placed into the stringList. The next time this call is performed, you may get an entirely different set of print queue jobs.

Example

```
var
   tempList : TStringList ;
   queueJobID : TNWQueueJobID ;
begin
         tempList := TStringList.create ;
         if getQueueJobList(0,'laser1',tempList) then
                listbox1.items := tempList ;

        {here's how to get queue job ID from the first stringlist element}
        queueJobID := TNWQueueJobID(tempList.objects[0]) ;
 end;
```

See Also

getQueueJobNumbers,
getQueueJobInfo

getQueueJobNumbers : boolean

Retrieves a listing of all queue job numbers in a specified print queue at that instant in time. Each time this function is called, it may return a different listing of print queue job ids, depending on the state of the jobs of the queue in question.

Parameters

nServer: TNWConnHandle. The file server handle that owns the print queue to query.

QueueName: string. The name of the print queue in which to query.

QueueJobList : TNWQueueJobList. A zero-based array of print queue job IDs. You iterate through this list until you obtain a job ID of 0, or you pass the list's highest offset, which is 255.

Returns

Boolean. True if this queue is read properly and joblist array is filled with the queue jobs IDs. If there are no print jobs in the queue, the function returns true, but the first element in the joblist is 0.

Example

```
var
  jobList : TNWQueueJobList ;
begin
  if getQueueJobNumbers(0,'laser1',jobList) then
     begin
            ntemp := 0 ;
            while (jobList[ntemp] > 0) do begin
                printLn('Job ID: ' + intToStr(jobList[ntemp])) ;
                inc(ntemp) ;
            end;
     end;
end;
```

See Also

getQueueJobList
getQueueJobInfo

getQueueOperators : TStringList

Use this function to obtain a complete listing of the object names that have been specified as valid Print Queue Operators. Queue Operators can hold, delete and get information on a given print queue's job entries, even if they do not own them.

It's usually a good idea to grant Queue Operator status to a Group Name that is physically located near the printers, or to those that often send jobs to the printer. Otherwise, they'll constantly bother the system administrator to manage the print queue for them when someone sends 500 copies of a job and then logs out.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

queueName: String. The name of the print queue to query.

Returns

TStringList. The output of this function can easily be incorporated into other VCL objects that contain a TStringList property, such as TListBox.

See Also

getQueueUsers
getQueueServers

getQueueServers : TStringList

Use this function to return a list of print servers that is currently directing output to a particular print queue. A user must be a QueueOperator in order to retrieve the names of the print servers submitting jobs to the print queue.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cQueue: String. The name of the print queue to query.

Returns

TStringList. A complete listing of the print servers submitting print jobs into the queue.

Example

```
listbox1.items.addStrings(getQueueServers(0,'PRINTER1')) ;
```

See Also

getQueueUsers
getQueueOperators

getQueueUsers : TStringList

This function queries a particular PrintQueue object, and returns the names of the user objects contained in the Queue Users set property. You can use the output of this function to verify a user can actually print to a specific print queue.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cQueue: String. The name of the print queue to query.

Returns

TStringList. The output of this function can easily be incorporated into VCL objects that contain a TStringList property, or you can iterate through the list yourself to locate a particular object name.

A list containing 0 entries is returned if the function call fails.

Example

```
listBox1.items.addStrings(getQueueUsers(0,'PRINTER1')) ;
```

See Also

getQueueOperators
getQueueServers

getServerDate : TDateTime

When you need to ensure a standardized date, use the getServerDate function. The return value of this function is not affected by the workstation's date. Good uses for this function are writing dates to time-sensitive data files that could be forged by changing the workstation's date using DOS' DATE command.

Parameters

nServer : TNWConnHandle. The connection handle of the server you want to query. You must be logged into the file server to perform this function.

Returns

TDateTime. The date of the file server. You can use any standard Delphi date functions to parse the return value.

Example

```
if getServerDate <> date then
   okBox('Please Stop Resetting Your PC Clock!') ;
```

See Also

getServerTime, setServerDateTime

getServerHandle : TNWConnHandle

Use getServerHandle when you know the physical name of a file server, and wish to retrieve it's connection handle.   Server connection handles are used quite frequently in NWLib, because they define precisely which server on the network to perform various function calls.

A handy use for this function is iterating though a getConnectedServerList return list to perform actions on the various servers.

Parameters

cServer : String.   The name of the server you want to inspect.

Returns

TNWConnHandle.   The connection handle to the specified server.   0 if the server name is not valid.

Example

```
var
   nServer : TNWConnHandle ;
begin
  nServer := getServerHandle('FS1') ;
  if nServer > 0 then
    begin
       OKBox('You Are Logged Into FS1 Properly') ;
    end;
end;
```

See Also

getServerName, getConnectedServerList, sList, isLoggedIn, NWLogin

getServerHandleFromPath : TNWConn ;

Betcha someday you'll want to find out the name of the server in which a drive mapping is based. With this function, you simply pass along a complete pathname, and it returns back to you the calling workstation's server connection ID of the host server of that drive mapping.

Once you have obtained a server's connection handle, you can perform just about any Netware call relating to that server, even if it's not your default or preferred server connection handle.

Parameters

cPath : string. The complete pathname you want to query. For instance if , 'g:\' is a valid pathname on your network, it is a valid parameter for getServerHandleFromPath.

Returns

nConnHandle : TNWConnHandle. The server connection handle.   On local or invalid pathnames, 0 is returned.

Example

```
var
  nConnHandle : TNWConnHandle ;
begin
  nConnHandle := getServerHandleFromPath('g:\') ;
  setPreferredServer(getServerName(nConnHandle)) ;
end;
```

See Also

setPreferredServer, all bindery/login/server functions.

getServerName : string

Use GetServerName to retrieve the name of a file server. The calling workstation must be logged into the file server to be able to perform this function.

Parameters

nServer: TNWConnHandle. The connection handle to the file server.

Returns

string: The name of the file server.

Example

```
var
   cserver: string
begin
   cserver := getServerName(getPrimaryServerID) ;
   okBox('You Are Connected to Server ' + cserver) ;
end;
```

See Also

sList, getConnectedServerList, getPrimaryServerID, getServerHandle

getServerSerial : longInt

Each file server on your network has it's own unique serial number. getServerSerial simply retrieves this number for you. Good uses for this number are encryption keys, security, and copy protection schemes.

a Netware serial number is obtained from the Netware System1 diskette in a standard Netware 3.1 installation, or from the CDROM in a Netware 4.x installation. You cannot change a file server's serial number.

Parameters

cServer : string. The name of the file server you wish to query. You can query any file server in which you have an active login.

Returns

longint: The file server serial number.

Example

```
var
  nserial : longint ;
begin
  nserial := getServerSerial('FS1') ;
  if nserial <> 39847763 then
    okBox('This is an Invalid Copy of WidgetWare!') ;
end;
```

See Also

getServerHandle, getServerName, getConnectedServerList, sList, NWLogin

getServerStats : boolean ;

With getServerStats, you're able to retrieve statistical information about any particular Netware 4.x server on your network.   With a little imagination and a few hours work, you can create network server monitoring and threshold alert systems.

Note: Some of the returned values in the reference structure are only applicable for Netware 4.x servers.

Parameters
nServer : TNWConnHandle. The server in which to query.

Returns
Boolean. True if the call was successful, and the referenced structure is filled with desired information.

Example
```
var
   serverInfo : TNWServerInfo
begin
   if getServerStats(getPrimaryServerID,serverInfo) then
         okBox('Server Ticks: ' + intToStr(serverInfo.serverUpTime)) ;
end;
```

See Also
getUserStats

getServerTime : TDateTime

Use the getServerTime function to retrieve a file server's time.   You can use this function to ensure a synchronized time is used throughout your program.

Parameters

nServer : TNWConnHandle.   The server connection to query.   You must be logged into the specified server to make this call.

Returns

TDateTime.   A standard Delphi TDateTime return value, which can be parsed, decoded or any other time-related function.

Example

```
var
   nHour : word;
   nMin   : word ;
   nSec   : word ;
   nMsec : word ;
   ntime : TDateTime;
begin
   ntime := getServerTime(getPrimaryServerID) ;
   DecodeTime(nTime,nHour,nMin,nSec,nMsec) ;
   if nHour = 7 then
      begin
          MessageBeep(mb_Exclamation) ;
          OKBox('Wake Up!   It's 7am You SleepyHead!') ;
      end;
end;
```

See Also

Paragraph

getTrusteeList : TStringList

Some objects, such as Users and Groups, can contain trustee rights which allow access info specific directories and/or files. You can obtain a complete listing of the trustee rights assigned to any object on the network with a single call to this function.

You must have Bindery Read Access Level of BS_SUPER_READ to the object in order to obtain a listing of the object's trustee list.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object in which the trustee list is obtained. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objectName: String. The name of the object in which the trustee rights list is obtained.

Returns

TStringList Each of the trustees and associated trustee rights are returned to you as an easy-to-use TStringList object. This enabled the use of the output of this function as the basis for TListBox.items, or you can iterate easily through the list yourself using the inherent TStringList object properties and methods.

An empty stringList is returned if the call fails, or the specified object contains no trustee rights.

Example

```
var
  cpath : string ;
  ntemp : word ;
  tempList : TStringList ;
begin
  result := false ;
  cpath := 'SYS:PUBLIC'
  tempList := getTrusteeList(  getServerHandle('FS2'),
                               'SUPERVISOR') ;
  for ntemp := 1 to tempList.count do begin
    if (pos(cpath,tempList[nloop-1])>0) then
       begin
           result := True ;
           okBox('SUPERVISOR Has Trustee Rights to SYS:PUBLIC!') ;
           break ;
       end;
   end;
end;
```

See Also

getEffectiveRights
getObjectDirRights
getPropertyList

modifyTrusteeRights
deleteTrusteeRight

getUserConnList : boolean ;

When you need to find out how many times a user is logged in, and the logical connection numbers, getUserConnList is the function you'll use to get the information.

Parameters

nServer : TNWConnHandle.   The server in which to query.

cUserID : string.   The name of the user whose connection status you are obtaining.

var nConns: word.   The number of connections in use by the specified cUserID is returned in this referenced variable upon successful completion of the function call.

var connList : TConnList.   The connection numbers in use by the specified cUserID is returned in this referenced array of connection IDs.   Each element in the returned list is of type TNWConnNumber.

Returns

Boolean. True if the call is successful and the referenced items are filled properly.

Example

```
var
   nconns     : word ;
   connList : TConnList ;
   nloop         : word ;
begin
   if getUserConnList(getPrimaryServerID,'JIM',nconns,connList) then
         begin
             for nloop := 1 to nconns do begin
                   okbox('Found Connection Number: ' + intToStr(connList[nloop-1])) ;
             end;
         end;
end;
```

See Also

getUserStats

getUserStats : boolean ;

Under Netware 4.x, you can call getUserStats to return user input/output statistics and other connection information.   You must be a console operator to view statistical information about a connection.

Parameters

nServer : TNWConnHandle.   The host server connection handle.

nConn: TNWConnNumber.   The user's connection ID.   Obtain it using GetUserConnList..

var connStats : TNWConnStats.   A referenced structure filled with values on successful completion of the call.

Returns

Boolean.   True if the call is completed successfully.

Example

```
var
   connStats : TNWConnStats ;
   connList     : TConnList ;
   nconns        :   word ;
   nloop             : word;
begin
   connList := getUserConnList(getPrimaryServerID,'JIM',nConns,connList) ;
   for nloop := 1 to nConns do begin
           if getUserStats(getPrimaryServerID,connList[nloop-1],connStats) then
                 okBox('Bytes Read: ' + intToStr(connStats.bytesRead)) ;
   end;
end;
```

See Also

getServerStats, getUserConnList

getVolFileList : TStringList ;

Use getVolFileList to retrieve the contents of a network volume.

Parameters

srchText : string.   The full path and file specification in which to search.   Wildcards are allowed.

Returns

TStringlist.   The contents of the specified path and file specification.   You can easily incoprorate the output of this function into your listbox and stringGrid objects.

Example

```
var
   tempList : TStringList ;
begin
   templist := TStringList.create ;
   templist := getVolFileList('z:\public\*.*') ;
   listbox1.items.addStrings(templist) ;
end;
```

See Also

getFileInfo, getDeleteFiles, getVolumes

getVolumes: TStringList ;

getVolumes is a convenient method to obtain a listing of available server drive volumes. Since the return value of this function is a TStringList, you can easily incorporate the information into your ListBoxes or StringGrids.

Parameters

nServer: TNWConnHandle. The server connection ID.

Returns

TStringList. The listing of available mounted volumes on the specified server.

Example

```
begin
  listbox1.items.addStrings(getVolumes(getPrimaryServerID)) ;
end;
```

See Also

Volinfo, getVolFileList, getFileInfo, getDeletedFiles

incSemaValue : boolean

Increments the arbitrary value of the specified semaphore. Each semaphore contains an arbitrary value that you can increment and decrement at will, to control whatever aspect of the environment desired.

Parameters

nServer: TNWConnHandle. The server that owns the semaphore to increment. Pass a zero and NWLib defaults to the current server.

SemaHandle: The semaphore handle previously obtained with a call to createSemaphore.

amount: The amount to increment the semaphore value. The semaphore value can contain from 0 to 127.

Returns

Boolean. True if the value was incremented. False indicates an invalid semaphore handle or no additional values are available.

Example

```
if incSemaValue(0,semaHandle,10) then
   ok('Incremented 10 times') ;
```

See Also

createSemaphore
decSemaValue

isCaptured : boolean

isCaptured can tell you if a particular LPT port on the workstation has been redirected to a network print device.

Parameters

nPort : TNLpt.   The LPT Port to check.

Returns

Boolean.   True if the port is redirected somewhere on the network.

Example

menuEndcap.enabled := isCaptured(1) ;

See Also

Capture, EndCap, setCaptureFlags, endCaptureFlags

isConsoleOperator : boolean

Query logged-in user's status to see if they have Console Operator rights. You might call this before attempting such functions as File Server statistics, LAN I/O information, or anything else that a regular Joe user should not be able to see.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

Returns

Boolean value. True if the logged in User is a console operator on the given server.

Example

```
menuStats.enabled := isConsoleOperator(GetPrimaryServerID) ;
```

isInList : boolean

isInList simply checks to see if a particular user exists within a comma-delimited string. isInList is a lot like IsMember. However, IsMember checks only one user against one group at a time, and isInList compares a user against many users and/or groups in a comma-delimited string all at once. It's just like if you created your own loop and called IsMember on each element of a comma-delimted list, skippnig anything that was not a group or did not equal the userID itself (which in fact is exactly what IsInList does!).

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : string. The UserID you want to test

GroupList : string. A comma-delimited list of users and/or groups that serve as the source data to test against.

Returns

True if cUserID exists anywhere in GroupList, including membership in one or more of any specified groups.

Example

```
menuLotus.enabled := isInList(0,WhoAmI(0),'BOB,JOE,ACCOUNTING,LOTUS') ;
```

isLoggedIn : boolean

IsLoggedIn can tell you if a particular user is logged into a file server in which you are connected. This is handy when you'd like to build a UserList of only those users that are actually logged into the network.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : string. The UserID you want to test, like SUPERVISOR or JOE.

Returns

Boolean. True if the specified user is logged into the network.

Example

```
if isLoggedIn('SUPERVISOR') then
   SendLineMessage('SUPERVISOR','You Are a Nerd!') ;
```

isMember : boolean

Use this function when you need to know if a user is a member of a particular group. For instance, you may have a need to restrict certain menu items to those users that are members of a particular group. This function can be called directly to set a menu item's .enabled property, thereby disabling the item for users which are not members of the specified group.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : string. The UserID you want to test. Like SUPERVISOR.

cGroup: string. The Group Name you want to validate against. Like EVERYONE.

Returns

Boolean. If cUserID is a member of the group specified in cGroup, TRUE is returned, FALSE if not. If the UserID or Group is invalid, False is returned.

Example

```
menuPack.enabled := IsMember(WhoAmI,'SUPERDUPER') ;
```

isNWManager : boolean

Use IsNWManager to find out if a user has been defined as a Netware Manager. You can specify users as managers in the Novell SysCon and NetAdmin programs.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

Returns

Boolean: True if the logged in user is a Netware Manager. False if not.

Example

```
menuSuper.enabled := isNWManager ;
```

longSwap : longInt

Swaps the high-and-low order words in a longint integer type. Since Netware stores most longints in reverse order, you'll need to use this function if you make any Netware API calls directly, and a longint data type is returned from the API call.

Parameters

longNumber : LongInt   This is the longint data type to reverse.

Returns

longInt. The given longint, only swapped.

Example

```
var
  longnumber:   longint ;
begin
   longnumber := longSwap(1837837938) ;
end;
```

Map : boolean

Use the Map function when you want to create a drive mapping 'on the fly' within your application. You pass along a drive letter and complete netware-compatible path string, and NWLib allocates the drive for you and adds it to the available system drives.   You can then access the data contained in the path as if you had mapped the drive using standard Netware commands.

Parameters

cDrive : char.   The drive letter you want to create.
cPath : string.   The complete path specification, including server if desired.

Returns

boolean: True if the operation was successful and the drive mapping is properly created.

Example

```
if map('g','fs1/sys:mail') then
   okbox('Drive Map Created Successfully');
```

See Also

GetNextNetDrive

mapDelete : boolean

Use the mapDelete function when you need to complete remove a Netware drive mapping for the system table of available drives.

No error checking or validation is performed, so your application should provide the means to close any open data files that may exist on the removed drive.

Parameters

cDrive : char.   The drive letter to remove.

Returns

Boolean: True if the drive is removed successfully.

Example

```
if MapDelete('g') then
   OKBox('Drive G Removed') ;
```

See Also

Map, MapShow, GetFirstNetDrive

mapShow : TStringList

Use mapShow when you need to create a parseable listing of all available drives on the workstation.

Parameters

None.

Returns

TStringList: A parseable listing of all available network and local drives.   The first two characters in each stringlist element represent the drive letter and colon.   The remainder of the data represents the complete path.

Example

```
var
   tempList : TStringList ;
begin
   templist := TStringList.Create ;
   tempList.Addstrings(mapShow) ;
end;
```

maxConns : TNWNumber

This function returns the maximum number of configured server connections on the workstation.   By default, Netware allows up to 8 server connections under an IPX/ODI Driver setup.   Under VLMs, up to 32 server connections can be configured by modifications to the NET.CFG file.

See your network documentation for more information about your NET.CFG file.

Parameters

None

Returns

TNWNumber: The total number of connections supported on the workstation.

Example

```
var
  tempList : TStringList ;
begin
  for nloop := 1 to maxConns do begin
      tempList.Add('Server Entry') ;
  end;
end;
```

modifyTrusteeRights : boolean

Use this function to change an object's directory trustee rights on a particular server. You must be a SUPERVISOR to modify an object's directory trustee rights.

You can add new trustee rights or modify existing rights with this function.

Parameters

nServer : TNWConnHandle The server connection handle that owns the object in which directory trustee rights are to be modified.

Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserName : String The name of the object in which a trustee right is added or edited. This can specify any Netware object that can have directory trustee rights, such as users and groups.

cVolName : String The name of the drive volume that contains the directory in which the trustee rights exist, or are to be created. You may pass an empty string if the complete volume and path information is specified in the cPath parameter.

cPath : String The complete pathname to the directory in which trustee rights are added or modified. The path must exist. If you pass an empty string in the cVolName parameter, you must include the volume name as part of the directory pathname.

Returns

Boolean. True if the specified trustee right was added or modified for the particular object. False usually means the object specified is not valid, or the path does not exist.

Example

```
var
   rightsList : TNWRightsList ;

with rightsList do begin
   supervisor := false ;
   read := true ;
   write := true ;
   create := true ;
   erase := true ;
   modify := true ;
   fileScan := true ;
   accessControl := false ;
end;

if modifyTrusteeRights(
         getPrimaryServerID,
         'GUEST',
         'sys:',
         'public',
        rightList) then
   okbox('Operation Completed Successfully!') ;
```

See Also

deleteTrusteeRight
getEffectiveRights
getObjectDirRights
getPrimaryServerID
getServerHandle

Sub Heading

Paragraph

nCopy : boolean

nCopy is actually a high-level interface to two seperate functions.   The first of which is an ultra-high-speed server-to-server copy function which works only on files that exist on the same file server. The next is for all other files, and is not quite as fast as the first, especially if the files in question reside over network connections vs. completely local storage.

nCopy first checks to see if the source and target files exist on the same file server,   If so, the function simple informs Netware the file needs to be copied, then returns the API calls returns value. Since the file is copied mostly into memory, and the file contents never travel down the network wire, the copy is usually instantaneous on average performing networks.

nCopy automatically reverts to the standard copy routine if either of the two input files do not reside on the same physical file server storage.

Parameters

cSourcefile : String.   The full DOS pathname of the file you want to copy.

cTargetFile : String.   The complete DOS target path of the destination file.   The directory must already exist.   If the file already exists, it will be overwritten without warning.

Returns

Boolean.   True if the file was copied successfully.

Example

```
if nCopy('g:\home\giant.xls','h:\excel\data\giant.xls') then
   OKBox('File Copied OK!') ;
```

ndsAbbreviateName : string

Converts a Directory Name (including all attribute type specifications) to the shortest form relative to the current context.

Parameters

directoryName : String

Returns

String.   The shortest form of directoryName relative to the current context.

Example

```
edit1.text := ndsAbbreviateName(ndsGetContextName) ;
```

See Also

ndsExpandName
ndsGetRootName

ndsClose : boolean

Call this function when you are ready to shut down an application which uses NDS services. This call releases the memory allocated for Novell's Unicode tables by NWLib's NDSInit call.

Parameters

None.

Returns

Boolean. True if the allocated memory is released. False usually indicates the NDS Environment was not properly initialized in the first place.

Example

```
if (not NDSClose) then
      alertBox('Error Releasing NDS Unicode Table Allocation!') ;
```

See Also

NDSInit

ndsCopyContext : TNWDSContextHandle

Creates a new context for NDS client operations and initializes it to the same settings as the context in whose handle it is passed.

Parameters

contextHandle : TNWDSContextHandle The source context handle to duplicate. A clone of the contextHandle is created that the client may access, change and call completely independent of the source contextHandle.

Returns

TNWContextHandle. The cloned context handle. 0 is returned if the operation could not be completed successfully.

Example

```
var
  newContextHandle : TNWDSContextHandle ;
  newContextName   : String ;
begin
  newContextHandle := ndsCopyContext(ndsGetContextHandle) ;
  if (newContextHandle > 0) then
    begin
       newContextName := getContextName(newContextHandle) ;
       okbox('Duplicate Context Handle Created!') ;
    end;
end;
```

See Also

ndsGetContextHandle
ndsSetContextName

ndsExpandName : string

Returns the complete context name of an abbreviated directory context name, relative to the active context.

Parameters

directoryName : String. The abbreviated name to expand.

Returns

String. The complete name of directoryName relative to the current context.

Example

```
edit1.text := ndsExpandName(ndsAbbreviateName(ndsGetContextName)) ;
```

Sub Heading

Paragraph

ndsFreeContext : boolean

Release memory allocated for context handle.   Memory is allocated for the context handle by such function calls as ndsInit and ndsCreateContext.   You should always ensure this allocation is freed before your program terminates if possible.

Parameters

contextHandle : TNWDSContextHandle	The context handle to free.

Returns

Boolean.   True if the contextHandle allocation is released.   False usually indicates the contextHandle does not exist.

When a context handle is freed, the context is no longer available at the workstation.   Any attempts to gather information about this context will fail.

Example

```
if ndsFreeContext(ndsGetContextHandle) then
    close
else
    alertBox('Error Closing NDS Context Handle!') ;
```

See Also

ndsGetContextHandle
ndsCreateContext
ndsGetContext
ndsSetContextName

ndsGetAttrRights : TNWDSAttrRights

Returns a summary of an object's effective attribute rights with respect to a given object in the current context.

Parameters

sourceObject : string.   The name of the source object in which attribute rights are obtained.

targetObject: string.   The name of the target object in which attribute rights of the sourceObject are compared.

VAR attrRights : TNWDSAttrRights.   The predefined record structure containing the attribute rights sourceObject is granted in respect to targetObject.

Returns

Boolean. True if the attribute rights are successfully retrieved. False if the call fails or the given objectName(s) do not exist in the current context.

Example

```
var
  attrRights : TNWDSAttrRights ;
begin
   if ndsGetAttrRights(ndsWhoAmI,'bravo.admin.john',attrRights) then
     begin
        okBox('Your Attribute Rights:;' +
                    iif(attrRights.compare,'Compare;','') +
                    iif(attrRights.read       ,'Read;'     ,'') +
                    iif(attrRights.compare,'Write;'      ,'') +
                    iif(attrRights.compare,'Self;'        ,'') +
                    iif(attrRights.compare,'Supervisor','') ) ;
    end;
end;
```

See Also

ndsGetSMSRights
ndsGetEntryRights

ndsGetBinderyContextName : string

If your environment contains workstations making calls to a Netware 4.x server in Bindery Emulation mode, you'll probably need this function to determine the workstation's location in the server's context.

Parameters

nServer: TNWConnHandle.   The server's connection handle in which to retrieve the bindery context name.

Returns

String.   The resolved bindery context name.   This is the entry point into the NDS tree the workstation is seeing as root of an emulated bindery.

Example

```
edit1.text := ndsGetBinderyContextName(getPrimaryServerID) ;
```

See Also

ndsGetContextName
ndsSetContextName
ndsGetRootName
ndsGetContextHandle

ndsGetContextHandle : TNWDSContextHandle

This function returns the active context handle in use at the current workstation.   This handle can be used in other NWLib functions, in in direct Netware API calls.

Parameters
None.

Returns
TNWDSContextHandle.   The current context's handle.

Example
```
var
   contextHandle : TNWDSContextHandle ;
begin
   contextHandle := ndsGetContextHandle ;
end;
```

See Also
ndsFreeContext
ndsCopyContext

ndsGetContextName : string

Returns the name of the current Directory Services context handle in use at the workstation.

Parameters
none.

Returns
String. The name of the Directory Context as a string value.

Example
```
edit1.text := ndsGetContextName ;
```

See Also
ndsSetContextName
ndsGetContextHandle
ndsFreeContext

ndsGetEntryRights : boolean

Returns a summary of an object's effective entry rights to a given object in the current context.

Parameters

sourceObject : string.   The name of the source object in which Entry rights are obtained.

targetObject: string.   The name of the target object in which Entry rights of the sourceObject are compared.

VAR entryRights : TNWDSEntryRights.   The predefined record structure containing the entry rights sourceObject is granted to targetObject.

Returns

Boolean.   True if the entry rights are successfully retrieved.   False if the call fails or the given objectName(s) do not exist in the current context.

Example

```
var
  entryRights : TNWDSEntryRights ;
begin
   if ndsGetEntryRights(ndsWhoAmI,'bravo.admin.john',entryRights) then
     begin
        okBox('Your Entry Rights:;' +
                     iif(entryRights.browse,     'Browse;'          ,'') +
                     iif(entryRights.add          ,'Read;' ,'') +
                     iif(entryRights.delete,      'Delete;','') +
                     iif(entryRights.rename,     'Rename;','') +
                     iif(entryRights.supervisor,'Supervisor','') ) ;
    end;
end;
```

See Also

ndsGetAttrRights
ndsGetSMSRights

ndsGetObjID : TObjID

Returns the raw Object ID number of a user relative to the current directory context.

Parameters

nServer: TNWConnHandle.   The server connection handle to query.

userName : String.   The complete context name of the user, or the relative name of the user in which to obtain the Object ID.

Returns

TObjID.   The raw Object ID of the given userName.   The return value can be converted to Hexidecimal to obtain the user's Hex Object ID, or passed through getObjName to retrieve the user's current context name.

Example

```
var
  rawObjID : TObjID ;
begin
  rawObjID := ndsGetObjID(   getPrimaryServerID,
                             'bravo.admin.jim') ;
  okBox('User Name: ' + getObjName(rawObjID) +
            ';User Object ID: ' + intToHex(rawObjID) ) ;
end;
```

See Also

getObjName

ndsGetRootName : string

Returns the partition root name of any given object.

Parameters

objectName : string.   The name of the object to query.

Returns

String.   The resolved partition root name of the given objectName.   An empty string is returned if the function call is not successful, or the objectName does not exist in the current context.

Example

```
edit1.text := ndsGetRootName('bravo') ;
```

See Also

ndsExpandName

ndsGetServerDN : string

Returns a Netware 4.x server Distinguished Name.

Parameters

nServer : TNWConnHandle The server connection handle in which to retrieve the distinguished name.

Returns

The file server's complete distinguished name as a string type.

Example

```
edit1.text := ndsGetServerDN(getPrimaryServerID) ;
```

See Also

ndsGetBinderyContextName
ndsGetContextName
ndsGetRootName

ndsGetSMSRights : TNWDSsmsRights

Returns a summary of an object's effective SMS rights with respect to a given object in the current context.

Parameters

sourceObject : string.   The name of the source object in which sms rights are obtained.

targetObject: string.   The name of the target object in which sms rights of the sourceObject are compared.

VAR smsRights : TNWDSsmsRights.   The predefined record structure containing the sms rights sourceObject is granted in respect to targetObject.

Returns

Boolean.   True if the SMS rights are successfully retrieved.   False if the call fails or the given objectName(s) do not exist in the current context.

Example

```
var
  smsRights : TNWDSsmsRights ;
begin
   if ndsGetSMSRights(ndsWhoAmI,'bravo.admin.sys1:',smsRights) then
     begin
        okBox('Your SMS Rights:;' +
                     iif(smsRights.scan,       'Scan;'          ,'') +
                     iif(smsRights.backup    ,'Backup;'      ,'') +
                     iif(smsRights.restore     ,'Restore;'     ,'') +
                     iif(smsRights.rename    ,'Rename'      ,'') +
                     iif(smsRights.delete       ,'Delete;'        ,'') +
                     iif(smsRights.admin       ,'Admin'          ,'') ) ;
    end;
end;
```

See Also

ndsGetAttrRights
ndsGetEntryRights

NDSInit : boolean

Before calling any of NWLib's NDS functions,   you should first verify the workstation and server have been properly initialized for NDS calls.

You can validate the presence of an NDS environment by making this call at the top of your program or unit.

Parameters
None.

Returns
Boolean.   True if the NDS environment is available.   False usually indicates the NDS environment is not present, or the user does not have the VLM shells loaded.

This function also initializes and allocates a memory buffer for Novell's Unicode tables.   This buffer must be freed on shutdown of the application with a call to NDSClose.

Example
```
if (not ndsInit) then
   begin
         alertBox('NDS Not Initialized Properly!') ;
         close;
   end;
```

See Also
NDSClose
NWInit

ndsLogin : boolean

Logs a workstation using VLMs running in NDS Mode into a 4.x Network.

Parameters

userName :	string.	The name of the user to log into the server.
passWord:	string.	The user's password.

Returns

Boolean. True if the user is successfully logged into the network.

Example

```
if (not ndsLogin('bravo.admin.jimt','myPassword')) then
   alertBox('Login Error!') ;
```

See Also

ndsLogout
NWLogin
NWLogout

ndsLogout : boolean

Logs a user out of a Netware 4.x network.

Parameters

none.

Returns

Boolean.   True if the logout is successful.

Example

```
if (not ndsLogout) then
   alertBox('Error Logging Out!') ;
```

See Also

ndsLogin
NWLogout
NWLogin

ndsPassCheck : boolean

Validate a user's Login Password on a Netware 4.x network.   Good for screen savers, workstation lockup programs, menu security, etc.

Netware always stores passwords in double-secret encrypted mode.   You cannot make any Netware API calls that will retrieve this password in textual, unencrypted form.

Parameters

userName : String.   The name of the user to validate.
Password : String.   The user's password to validate.

Returns

Boolean.   True is the userName/Password combination is correct.

Example

```
if ndsPassCheck('bravo.admin.jim','superFly') then
   begin
      {do something}
   end;
```

See Also

nwPassCheck

ndsSetContextName : TNWDSContextHandle

Use this function to change the active context on the current workstation.

Parameters

contextName : String.   The context name on the network in which to active as the current context.

You should free the context handle (ndsFreeContext) associated with this new context handle when you have finished with it.

Returns

contextHandle: TNWDSContextHandle.   The handle to the new context.   0 is returned if the call is not successful.

Example

```
contextHandle := ndsSetContextName('office.sales.texas') ;
if (contextHandle < 1) then
   alertBox('Specified Context does Not Exist!') ;
```

See Also

ndsGetContextName
ndsGetContextHandle
ndsFreeContext

ndsWhoAmI : string

Returns the caller's full directory name, relative to the context in use.

Parameters
none.

Returns
String. The user's full directory name, relative to the active context.

Example
```
edit1.text := ndsWhoAmI ;
```

See Also
whoAmI
ndsLogin

NWDateTimeToTDateTime : TDateTime

Converts a Novell Packed Date/Time format into a Delphi TDateTime type.

Mostly used internally, but you can call this function yourself if you are making any direct Netware API calls that contain a packed date/time value.

Parameters
inDate: TNWDateTime

Returns
TDateTime: A regular Delphi TDateTime type that can be used like any other TDateTime variable.

Example
```
edit1.text := formatDateTime('mm/dd/yy',
                        NWDateTimeToTDateTime(nwDateVar)) ;
```

nwInit : boolean

NWInit should be the first NWLib call that your programs make.   You call this function to initialize the Netware DLL.   You get back a boolean value which indicates whether or not the user has a valid Netware shell loaded.

If the user does not have a valid Netware Shell environment, you should not make any network environment calls, for obvious reasons.

Returns
Boolean value.   True means network initialization occurred properly, and you can make network function calls.   False means you should not make any network calls.

Parameters
none

Example
```
begin
  if not NWInit then
       begin
            AlertBox('Netware Not Loaded!') ;
            close
       end;
end;
```

NWLogin : boolean

To establish a valid connection to a file server, you need to first make a connection to it with the NWLogin function.   You pass the server and user name, along with a password and object type, and Netware validates the password and logs the workstation into the specified server if all data is correct.

Parameters

cServer : String.   The name of the server in which to perform the login.   This server becomes the default server if a successful login is made.

cUserID : String.   The UserID in which to log in.   Like TODDR or JULIE.

cPassword : String. The Password to use.   NWLib encrypts the password down the wire if encrypted passwords are enabled.

nObjType : TObjType.   Netware allows other object types, such as Job Servers, Print Servers, etc. to log into the network.   This parameter allows you to specify   the object type (ie, nw_user, nw_group, etc.) to log into the server.   If you pass a 0 (zero) as this parameter, a user object (nw_user) defaults.

Returns

Boolean.   True if the login was successful.

Example

```
if NWLogin('fs1','ELAINER','hello',nw_user) then
   OkBox('Login OK!   Way to Go!') ;
```

NWLogOut : Boolean

Use NWLogOut when you want to completely sever a workstation's connection to a particular file server. The workstation is completely logged out, and the server connection handle is freed, making room for others if needed.

Use NWLogin if you want to re-attach to the file server and log in again.

Parameters

nServer: TNWConnHandle. The server handle you want to disconnect. No prompts or confirmations are displayed, so it's up to you to provide the necessary user interface/safety checks to assure the proper closing and saving of open data files.

Returns

Boolean. True if the logout was successful.

Example

```
if YesNoBox('Log Out of Server ' + GetServerName(getPrimaryServerID)) then
    NWLogout(GetPrimaryServerID) ;
```

nwPassCheck : boolean

nwPassCheck allows you to safely verify a user's Netware password. You simply pass a userID and password string, and NWLib checks it against the Netware Bindery/NDS to see if a match is made.

Netware stores it's password information for each user in an encrypted fashion. You can never retrieve the actual password from the Novell operating environment, only verify a password's validity. So, unless you have a user's password to start with, you cannot find out the password, even if you are a SUPERVISOR.

A common use for this function is password-protecting menu items or creating screen blankers or other security devices.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : string. The UserID who's password to validate.

cPassword : string. The password to validate.

Returns

Boolean: If cUserID's password really is cPassword, then True is returned.

False is returned if cCuserID is not a valid network user or the password is not correct.

- Note that Netware counts the number of attempts to validate a password, and may disable the login ID if too many incorrect attempts are made to try to validate a password. If your User account is disabled because of this 'intruder lockout,' you'll need to use your SysCon or NetAdmin utility to re-activate the account, or wait until the disarm time expires.

See Also

NDSPassCheck

parseNetwarePath : boolean

This function takes just about any format of path specifier, and converts it into a standard format which netware uses internally. From the referenced structure, you can obtain a lot of useful information, such as the server name that owns the path, the source volume ID, relative path, etc.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cPath: The fully-qualified path information, including drive letter or volume text information.

VAR pathInfo : TNWPathInfo

Returns

Boolean. True if the call is successful, and the given pathname actually exists on the specified server. In which case, the referenced TNWPathInfo record structure is filled with relative information.

Example

```
var
  pathInfo : TNWPathInfo ;
begin
  if parseNetwarePath(0,'z:\',pathInfo) then
     okBox('z: is really: ' + pathInfo.relativePath) ;
```

purgeAllFiles : boolean ;

Completely wipes out all recoverable files on a particular server without any confirmation or warning at all.

Once executed by a user with Console Operator rights, all deleted files are completely wiped out by the server.

Parameters
nServer : TNWConnHandle. The server connection handle in which to wipe out the deleted files.

Returns
Boolean. True if the operation was a success.

Example
```
if purgeAllFiles(getPrimaryServerID) then
    okbox('Purge Completed!') ;
```

See Also
Salvage, getDeletedFiles, getDeletedFileInfo

querySemaphore : word

Returns the number of workstations with the specified semaphore open. The current semaphore value is also returned by reference.

Parameters

nServer : TNWConnHandle. The server connection that owns the semaphore.

semaHandle: TNWSemaHandle. The semaphore handle to query.

var semaValue: word. The current semaphore 'value.' A semaphore can store any arbitrary value along with the station count for your own use.

Returns

word. The active stations with a lock on this semaphore. The semaValue parameter is updated to reflect the current semaphore arbitrary value.

Example

```
var
  semaHandle :  TNWSemaValue ;
  semaValue : word ;
begin
   semaHandle := createSemaphore(0,'NWLIB',10) ;
   if (semaHandle > 0) then
      okBox(intToStr(querySemaphore(0,semaHandle,semaValue) )  +
                  ' open connections to NWLib')
  else
     alertBox('Error Opening Semaphore!') ;
end;
```

See Also

createSemaphore

renameObject : Boolean

Use this function if you'd like to change the name of an existing Netware object. The object must already exist, of course.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be renamed. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

oldName: String The name of the existing object to rename.

newName: String The new name of the existing object.

objType: TObjType The object type, such as nw_user of nw_group.

Returns

Boolean. True if the operation is a success.

Example

```
if renameObject('BILLY','SALLY',nw_user) then
   okbox('Billy's Sex Change is a Success') ;
```

See Also

createObject, deleteObject

salvage : boolean ;

Use Salvage when you need to recover a file that's been deleted from a Netware volume.

Parameters

nServer : TNWConnHandle.   The server handle in which to recover the deleted file.

deletedFile : string.   The complete path specification and name of   the erased file.

outputFile: string.   The final output designation filename.   The file is written into the original file location.

Returns

Boolean.   True if the file is properly salvaged.

Example

```
if Salvage(getPrimaryServerID,'z:\public\deleted.doc','output.doc') then
   okbox('File Salvaged!') ;
```

See also

getDeletedFileInfo, getDeletedFiles, purgeAllFiles

SecureEquiv : boolean

Use SecureEquiv when you want to see if a user has a security level equal to or greater than another user on the network. A common use for this function is for SUPERVISOR security checks, ie, to enable certain menu items and such for those users which have SUPERVISOR equivalency.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object to be queried. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objectName : string. The Object Name you want to test, such as 'SUPERVISOR'.

cEquiv: string. The entity to be tested against, such as 'BOB'.

Returns

If UserID's security level is equal to or greater than cEquiv, True is returned. False otherwise.

Example

```
menuSuper.enabled := secureEquiv(WhoAmi,'SUPERVISOR') ;
```

sendLineMessage : boolean

Use sendLineMessage when you want to send a broadcast message to another user on the network. The target user must be logged into the network at the time, must not have Broadcast messages disabled, and must not have more than two messages already waiting in the server's broadcast queue (note: some third party utilities increase this 2 message limit by buffering the server's broadcast messages elsewhere other than the server's internal queue).

Parameters

nServer: TNWConnHandle The server connection handle that owns the object in which to send the message Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

cUserID : string. The target user in which to send the message. This user must be logged into the current server. If the user resides on a different server, make that server the default by using the SetPreferredServer function before calling sendLineMessage.

cMessageText: string. The text you want to send to the user. Note NWLib does not automatically pre-pend [Sender_UserID] to the front of the string. If you choose to leave this portion off the message, you in effect are sending anonymous messages.

Returns

Boolean: True if the message was delivered to the workstation successfully. False could mean a variety of things, including the user not being logged into the network, or having more than 2 messages already waiting at the file server.

Example

```
if SendLineMessage('Jim','[' + WhoAmI + ']    ' + Hello There!') then
  OKBox('Message sent to Jim!') ;
```

serverLoginOK   : boolean

Determines whether the server is accepting logins.   If the system administrator has 'Disabled Logins' or the server bindery is otherwise locked, this function lets you know.

Parameters

nServer: TNWConnHandle.   The server connection handle to query.   Pass a 0 (zero) as this parameter to specify the current, or default file server.

Returns

boolean.   True if network objects can access the server.   False may indicate a bindery locked condition.

If you suspect an invalid server connection handle, check the value of the lastError variable.   If it is non-zero, the call failed.

Example

```
checkBox1.checked := serverLoginOK(getPrimaryServerID) ;
```

See Also

NWLogin
NWLogout

setBannerUserName : boolean

If you need to change the UserName that appears on those print job banner pages, use this function. This is a seperate string from the banner text.

A Banner will not print at all if the Capture environment's BannerText is empty.

Parameters

cBannerName : String.   The UserName you wish to have printed on banners.

Returns

Boolean.   Successful if the Banner UserName was changed.

Example

```
setBannerUserName('JIMT') ;
```

See Also

getBannerUserName, Capture, setCaptureFlags, getCaptureFlags

setCaptureFlags : boolean

After a Capture has been performed on the workstation, you can use the setCaptureFlags function to change attributes about the capture environment already in place.   This change can happen right after a new NWLib Capture function call, or independently to leave the current Capture in place and just change it's attributes.

Parameters

nPort : TNWLpt.   The port number to alter.   Usually 1 to 3.

CaptureFlags : TNWCaptureFlags.   The predefined record structure containing the attributes that you want to change.

Returns

Boolean.   True if the Capture environment was changed to the new attributes specified in the the CaptureFlags record structure.

Example

```
var
  captureFlags : TNWCaptureFlags ;
  nLPT : TNWLpt ;
  oldqueue : string ;
begin
    nLPT := 1 ;
    if getCaptureFlags(nLPT,captureFlags) then
       begin
         captureFlags.formfeeds := False ;     { /nff }
         captureFlags.banner   := '' ;                { /nb }
         captureFlags.tabSize := 0 ;                { /nt   }
         oldqueue := captureFlags.qname ;     { i.e. Save old Capture, if you need }
         Capture('fs1','printer1',nLPT) ) then
           begin
                setCaptureFlags(GetPrimaryServerID,nLPT,captureFlags) ;
                okBox('Print Redirected to FS1/Printer1') ;
           end;
       end;
end;
```

See Also

getCaptureFlags, Capture, isCaptured

procedure setCastMode

This procedure attempts to change the broadcast mode at the current file server. This affects the way Netware broadcast messages are handled by the network shell (NETX or other VLM).

Parameters

nMode : specify one of the following predefined constants:

nw_caston :	enable all broadcasts; displayed immediately.
nw_castOff:	disable all broadcasts; discard them.
nw_castserver :	hold messages at server.

Returns

Nothing...it's a procedure, after all!

Example

```
if castoff.checked then
  setCastMode(nw_castOff) ;
```

Title

SetPreferredServer : boolean

Since Netware sometimes returns different information depending on which server is currently 'default,' you can use this function to change the active server. After this change, Print Capture, file server environment, bindery calls and other NWLib function calls start referring to this server connection.

Parameters

cServer : string. Name of the server you want to make default.
VAR nServer : TNWConnHandle. Passed by reference, NWLib returns back to you the server handle of the new default server. You might store this variable for future needs to call the current server handle.

Returns

Boolean. True if the change to the new server was successful. False if not. Maybe the cServer name is not valid?

Example

```
var
  templist : TStringList ;
  nloop : byte
  nConnhandle : TNWConnHandle ;
begin
  templist := TStringList.Create;
  templist.AddStrings(getConnectedServerList) ;
  for nloop := 1 to templist.Count do begin
     if setPreferredServer(templist[nloop-1],nconnHandle) then
       okBox('Connected To Server: ' + templist[nloop-1]) ;
  end;
end;
```

setQueueJobInfo : boolean

Changes details about a print queue job, such as the description, execution time, print status flags, etc.

Parameters

queueJobInfo : TNWQueueJobInfo.   A structure containing the server connection handle, print queue name, and queue job ID to change.   Change any other fields that you want to change about the print queue job.

You must supply at least the first three queueJobInfo[] elements in order to properly call this function.

Returns

Boolean.   True if the changes to the queue job were accepted.   The new queue job values are returned in the passed queueJobInfo structure.

Example

```
var
    queueJobInfo : TNWQueueJobInfo ;
    jobList : TNWQueueJobList ;
    serverConn : TNWConnHandle ;
    queueName : string ;
begin
   serverConn := getPrimaryServerID ;
   queueName : 'laser' ;
    if getQueueJobNumbers(serverconn,queueName,jobList) and
             (jobList[0] > 0) then
        begin
             queueJobInfo.nServer := serverConn ;
             queueJobInfo.cQueue := queueName ;
             queueJobInfo.jobID       := jobList[0] ;
             queueJobInfo.jobDescription := 'Changed Job Description' ;
             queueJobInfo.jobFlags.user_hold := true ;
             if setQueueJobInfo(queueJobInfo) then
                    okBox('Job Info Changed!') ;
        end ;
end;
```

See Also

getQueueJobInfo
getQueueJobList
getQueueJobNumbers

setQueueJobPosition : boolean

Changes the placement of a queue job within the current print queue. Other queue jobs are renumbered according to the selected position of the queue job moved.

To move a print job to the end of the queue, specify 250 as the new queue position, as Netware queues can only have 250 jobs in a queue at a time.

Parameters

queueJobInfo : TNWQueueJobInfo. A structure containing at least the server connection handle that owns the print queue, the print queue name, and the print job ID to manipulate.

You must supply at least the first three elements of the queueJobInfo structure in order to properly call this function.

Returns

Boolean. True if the call is successful, and the queue job is moved to the new postition. False usually indicates on of the first three parameters of the queueJobInfo structure do not point to a valid queue job

You cannot move a print job currently in service.

Example

```
var
   queueJobInfo : TNWQueueJobInfo ;
   jobList : TNWQueueJobList ;
begin
    queueJobInfo.nServer := getPrimaryServerID ;
    queueJobInfo.cQueue := 'laser'   ;
    if (not getQueueJobNumbers(queueJobInfo.nServer,
                                                   queueJobInfo.cQueue,
                                                    jobList)) or
        (jobList[0]=0) then
              exit ;
     queueJobInfo.jobID := jobList[0] ;
     if setQueueJobPosition(queueJobInfo,250) then
         okbox('Queue Job Moved to End') ;
end;
```

See Also

getQueueJobInfo
setQueueJobInfo
getQueueJobNumbers
getQueueJobList

setServerDateTime : Boolean

Pass setServer a server connection handle and a valid TDateTime value, and you'll instantly set the file server's date and time.

You must have console operator rights on the specified server in order to perform this function.

Parameters

nServer : TNWConnHandle.   The server whose time clock you wish to reset.

dt: TDateTime.   The new Date and Time.

Returns

Boolean.   True if the function was successful.   Next time you log into the file server (not attach), your PC Clock will be synchronized to this new time, unless you disabled that feature through a NET.CFG setting.

Example

```
var
  dt : TDateTime
begin
   dt := strToDateTime('12/25/95 12:00:01 am') ;
   if setServerDateTime(dt) then
      OKBox('Ho Ho Ho!   Merry Christmas!') ;
end;
```

See Also

Paragraph

sList : TStringList

Use the sList function when you need to fill up a TStringList with the names of the servers that are visible at the current workstation.   You can use this TStringList as the foundation for a listbox or create a loop and act on each of the server names independently.

Parameters

None

Returns

TStringList: The complete list of file servers advertising on your network.

Example

```
var
   templist : TStringList ;
   nloop : byte;
begin
   templist := TStringList.Create ;
   templist.AddStrings(sList) ;
   for nloop := 1 to tempList.count do begin
      okBox('Found Server: ' + templist[nloop-1]) ;
   end;
end;
```

TNWCaptureFlags

```
Copies              : longint    ;   # of Copies
TabSize             : longint    ;   Tab Conversion (0=No Tabs)
formType            : longint    ;   Form Number
timeOut             : longint    ;   Time Out Value
description  : string     ;   Job Name (For Queue Display)
formName     : string     ;   Form Name
banner              : string     ;   Banner Text (Empty=/NB)
qname               : string     ;   RO : Queue Name
nServer             : TNWConnHandle ;RO:   Server Name
autoEndCap   : boolean       ;       Auto-EndCap job on exit
formFeed            : boolean       ;       Enable FormFeeds?
JobNotify           : boolean       ;       Enable Job Notification?
```

TNWLib

nCopy
getFirstNetDrive
getNextNetDrive
parseNetwarePath
getPathInfo

longSwap

whoAmI : string

Just like Novell's own command line utility of the same name, WhoAmI allows you to fetch the current user's Login Name at the preferred server.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object in which to query. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

Returns

String: The name of the user logged in on the current server.

Example

```
begin
  footer.caption := whoAmI;
  if fileExists('m:\home\' + whoAmI(0) + '\zofo.zep') then
     alertBox('Cool.') ;
end;
```

NWNDS

Novell Directory Services Name and Context Management

```
{******** Public Record Structures ********}

type
  TNWDSAttrRights = record
       compare        : boolean ;
       read     : boolean ;
       write    : boolean ;
       self     : boolean ;
       supervisor : boolean ;
end;

type
  TNWDSEntryRights = record
       browse  : boolean ;
       add      : boolean ;
       delete   : boolean ;
       rename : boolean ;
       supervisor: boolean ;
end;

type
  TNWDSSMSRights = record
       scan     : boolean ;
       backup  : boolean ;
       restore  : boolean ;
       rename : boolean ;
       delete   : boolean ;
       admin    : boolean ;
end;
```

NDSInit
NDSClose
NDSGetServerDN
NDSGetBinderyContextName
NDSGetContextHandle
NDSFreeContext
NDSGetContextName
NDSCopyContext

NDSGetRootName
NDSAbbreviateName
NDSExpandName

NDSWhoAmI
NDSGetObjID
NDSLogin
NDSLogout
NDSPassCheck
NDSGetAttrRights
NDSGetSMSRights
NDSGetEntryRights

TNWPrint

TNWProp

Object and Property Manipulation Functions

TNWRights: General File and Directory Rights Structure

```
type
  TNWRights   =   record
        supervisor        : boolean ;
        read              : boolean ;
        open              : boolean ; {Netware 2.x Only}
        write             : boolean ;
        create            : boolean ;
        erase             : boolean ;
        modify            : boolean ;
        filescan          : boolean ; { Search in Netware 2.x}
end;
```

Used in: modifyTrusteeRights, getEffectiveRights, getObjectDirRights

writeItemProperty : boolean

A Netware object, such as a user or group contain a few BF_ITEM properties. Item properties can contain a single entry, such as the 'IDENTIFICATION' property, which holds the object's Full Name.

Use the writeItemProperty function to modify an existing property in a Netware object.

Parameters

nServer: TNWConnHandle The server connection handle that owns the object in which the new property is to be modified. Pass a 0 (zero) as the server handle and NWLib automatically uses the 'primary' server connection handle.

objName: String. Specify the name of the object that owns the Item property to be edited.

objType: TObjType. Use one of NWLib's predefined constants, such as nw_user or nw_group. NWLib's include file contains a complete listing of all valid Netware object types.

propName: String. Pass any existing property name that exists in the scope of the specified user. This function can modify any property, including new ones that you may have created yourself, or those created by Netware itself.

value: String. Specifies the replacement string that is written to the specified Object's property.

Returns

Boolean. True if the operation is successful and the object's property is modified. A return code of false usually indicates the server/user combination is invalid, or the object does not contain such a property.

Example

```
if writeItemProperty(getPrimaryServerID,
                  'SUPERVISOR',
                  nw_user,
                  'IDENTIFICATION',
                  'Techno-Jock') then
    okBox('Supervisor Full Name Changed!') ;
```

See Also

createProperty, createObject

TNWServer

volinfo : boolean ;

To obtain physical information about a Netware 2.2 server volume, use this function.

Parameters

nServer ; TNWConnHandle .   The Netware 2.2 server handle containing the volume in which to obtain physical information.

volName : string.   The physical volume name to query, such as 'SYS:' or 'VOL1:'

var volumeInfo : TNWVolumeInfo.   A referenced structure filled with all sorts of handy information.

Returns

Boolean.   True if the referenced structure is properly filled with physical volume information.

Example

```
var
  volumeinfo : TNWVolumeInfo ;
begin
   if volinfo(getPrimaryServerID,'SYS:',volumeInfo) then
      okBox('Total Blocks: ' + intToStr(volumeInfo.totalBlocks)) ;
end.
```

See Also

getServerStats

NWTools

String Manipulation

allTrim(ctext : string) : string ;
Removes all leading and trailing white space from a pascal-style string.

LTrim(ctext ; string) : string ;
Removes leading white space from a pascal-style string.

RTrim(ctext : string) : string ;
Removes trailing white space from a pascal-style string.

leftString(ctext : string) : string ;
Returns the left-most portion of a pascal-style string.

rightString(ctext : string; nLength: integer) : string ;
Returns the right-most portion of a pascal-style string .

padC(ctext : string ; nLength : integer ; cChar : char) : string ;
Add white space to both sides of a pascal-style string until it is x number of bytes in length.

Longer strings are truncated to the maximum length specified in nLength.

padL(ctext : string ; nLength : integer ; cChar : char) : string ;
Add white space to the left side of a pascal-style string until it is x number of bytes in length.

Longer strings are truncated to the maximum length specified in nLength.

padR(ctext : string ; nLength : integer ; cChar : char) : string ;
Add white space to the right side of a pascal-string until it is x number of bytes in length.

Longer strings are truncated to the maximum length specified in nLength.

Rat(ctoken : char ; ctext : string) : integer ;
Return the right-most ordinal position of a particular character within a pascal-style string.

strExtract(ctext : string ; cToken : char ; nPos : integer) : string ;
Extracts a delimited substring of a string. For example, you could use this function to extract out

the 4th element of a comma-delimted string.   You specify the sournce string, the characters which make up the delimiters, and finally, which sub-element you want extracted.

strTran(ctext, cFor, cWith : string) : string ;
Converts a all instances of a particular character in a string to any other character.

iif(Expression : boolean; cReturn1, cReturn2 : string) : string ;
Returns one of two strings, depending on boolean value.   Very handy for button captions and dynamic string handling functions.   For instance, you can build an expression that says "Yes" if the value is True, or "No" if the value is false.

Dialogs

procedure alertBox(cMessage : string) ;
Diaplay a standard modal Windows "Stopsign" dialog box.

procedure okBox(cMessage : string) ;
Displays a standard modal Windows "Information" dialog box.

yesNoBox(cMessage : string) : integer ;
Displays a standard modal Windows "Question" dialog box.

noYesBox(cMessage : string) : integer ;
Same as the YesNoBox, but with 'No' highlighted by default.

YNCBox(cMessage : string) : boolean ;
Displays a standard modal Windows "Yes/No/Cancel" dialog box.

Calculators

charCount(ctext : string; ctoken : char) : integer ;
Counts the occurences of a character within a string.

minLong(nval1, nval2 : Longint) : Longint ;
Returns the lower of two values.

maxLong(nval1, nval2 : Longint) : Longint ;

Returns the higher of two values.

delim2strList(ctext : string) : TStringList ;
Converts a comma-delimited string list into a TStringList object, filled with the values wthin the string. Great for converting data records into listboxes and such.

strEmpty(ctext : string) : boolean ;
True if the specified pascal-style's string length is less than 1.

aSub(atemp : array of char ; nstart, nend : word) : string ;
Returns a substring of a zero-based character array. You specify the starting and ending positions, and a pascal-style string is returned. Very handy for scanning large memos/pChars and displaying the contents in a button or other screen object.

stringListPos(strList : TStringList ; ctext : string ; var index : word) : boolean ;
Returns the ordinal position of a substring with a TStringList object. The value returned is the row in which the substring exists.

space(nlength : word) : string ;
Returns a string containing x number of white spaces.

between(nVal, nMin, nMax : longint) : longint ;
Tests to see if a value is between two other values.

File Functions

setPath(cpath : string) : string ;
Returns a nicely formatted path specification. It ensures the path contains a trailing backslash, the drive letter and colon are present within the string and it satisfies standard DOS naming conventions.

canOpen(cfile : string) : boolean ;
Tests to see if any particular file can be opened for read by your workstation. It's a good idea to do this before trying to open any file that may result in an undesired error message or situation.

funique(cpath : string) : string ;
Creates a guaranteed unique file name in the path you specify, and returns to you the name of the file that was created.

sizeofFile(cfile : string) : longint ;
Gives you the size of the specified file.

Date/Time Functions

timeTextInc(ctime : string; nway : integer) : string ;
Rounds time string up or down to the nearest 15 minutes. For instance, when incrementing a time string, 11:28 becomes 11:30 and 11:30 becomes 11:45.

dateTextInc(ctime : string; nway : integer) : string ;
Moves date string up or down by one day, honoring all standard leap year and last-day-of-month conventions.

formTime(cdate : TDateTime) : string ;
Returns a nicely formatted time string based on a TDateTime variable.

formDate(cdate : TDateTime) : string ;
Returns a nicely formatted date string based on a TDateTime variable.

lastDay(cDate : TDateTime) : byte ;
Returns the last day of the month of the given date. Takes into consideration all leap year activity, etc.

year(ctime : TDateTime) : byte ;
Extracts the year portion from a TDateTime variable.

month(cdate : TDateTime) : byte ;
Returns the month number from a date variable.

day(cdate : TDateTime) : byte ;
Returns the day portion of a TDateTime variable.

second(ctime : TDateTime) : byte ;
Returns the second portion of a TDateTime variable.

minute(cdate : TDateTime) : byte ;
Returns the minute portion of a TDateTime variable.

hour(ctime : TDateTime) : byte ;
Returns the second portion of a TDateTime variable.

Bindery Object Types

NWLib defines many Netware object types, including nw_user, nw_group, nw_printq, nw_server, etc.

createObject

<createObject>

createProperty

<createProperty>

TNWConnectInfo

<structures>

TNWConnStats

<structures>

TNWDeletedFileInfo

<structures>

TNWDiskCacheInfo

<structures>

TNWFileInfo

<structures>

TNWFileSysInfo

<structures>

TNWLib

<TNWLib>

TNWMemCacheInfo

<structures>

TNWNDS

<TNWNDS>

TNWPrint

<TNWPrint>

TNWProp

<TNWProp>

TNWRights
<structures>

TNWServer

<TNWServer>

TNWServerInfo

<structures>

TNWTools

<TNWTools>

TNWVolInfo

<structures>