Windows NT

Windows Networking APIs

Revision 1.0, August 14 1992

1. Overview 3

2. API buffers 3
   2.1. API Data Alignment 4
   2.2. Enumeration Buffer Lengths 4
   2.3. Parameter Error Reporting 4
   2.4. Parmnum For SetInfo 4
   2.5. Parmnum For GetInfo 4
   2.6. Embedded Strings 5
   2.7. Enumeration Resume Handles 5
   2.8. User-specific information 5
   2.9. Enumeration APIs 5
   2.10. SetInfo APIs 6

3. Windows Networking APIs 6
   3.1. API Status 6
   3.2. RPC Buffer Allocation Errors 6
   3.3. Obsolete Information Fields 6
   3.4. NLS Support 7

4. Windows Networking API Definitions. 7
   4.1. Buffer Manipulation APIs. 7
      4.1.1. NetApiBufferAllocate 7
      4.1.2. NetApiBufferFree 7
      4.1.3. NetApiBufferReallocate 8
      4.1.4. NetApiBufferSize 8
   4.2. Service APIs. 8
      4.2.1. NetServiceControl 9
      4.2.2. NetServiceEnum 10
      4.2.3. NetServiceGetInfo 11
      4.2.4. NetServiceInstall 12
   4.3. Server APIs 13
      4.3.1. NetServerEnum 20
      4.3.2. NetServerGetInfo 21
      4.3.3. NetServerSetInfo 22
      4.3.4. NetServerDiskEnum 23
   4.4. Workstation and WorkstationUser APIs 24
      4.4.1. NetWkstaGetInfo 32
      4.4.2. NetWkstaSetInfo 32
      4.4.3. NetWkstaUserGetInfo 33
      4.4.4. NetWkstaUserSetInfo 34
      4.4.5. NetWkstaUserEnum 34
   4.5. Use APIs 35
      4.5.1. NetUseAdd 37
      4.5.2. NetUseDel 37
      4.5.3. NetUseEnum 38
      4.5.4. NetUseGetInfo 39
   4.6. Share APIs 39
      4.6.1. NetShareAdd 41
      4.6.2. NetShareEnum 42
      4.6.3. NetShareGetInfo 43
      4.6.4. NetShareSetInfo 43
      4.6.5. NetShareDel 44

4.6.6. NetShareCheck 45
4.7. Session 45
4.7.1. NetSessionEnum 46
4.7.2. NetSessionGetInfo 48
4.7.3. NetSessionDel 48
4.8. Connection APIs 49
4.8.1. NetConnectionEnum 50
4.9. File APIs 51
4.9.1. NetFileEnum 52
4.9.2. NetFileGetInfo 53
4.9.3. NetFileClose 54
4.10. Message APIs 54
4.10.1. NetMessageNameAdd 55
4.10.2. NetMessageNameEnum 55
4.10.3. NetMessageNameGetInfo 56
4.10.4. NetMessageNameDel 57
4.10.5. NetMessageBufferSend 57
4.11. Remote Utility API 58
4.11.1. NetRemoteTOD API 58
4.12. Security Account APIs 59
4.12.1. User APIs 59
4.12.1.1. NetUserAdd 65
4.12.1.2. NetUserEnum 66
4.12.1.3. NetUserGetInfo 67
4.12.1.4. NetUserSetInfo 68
4.12.1.5. NetUserDel 68
4.12.1.6. NetUserGetGroups 69
4.12.1.7. NetUserGetLocalGroups 70
4.12.1.8. NetUserSetGroups 71
4.12.2. User Modal APIS 72
4.12.2.1. NetUserModalsGet 73
4.12.2.2. NetUserModalsSet 74
4.12.3. Global Group APIs 74
4.12.3.1. NetGroupAdd 76
4.12.3.2. NetGroupAddUser 77
4.12.3.3. NetGroupEnum 77
4.12.3.4. NetGroupGetInfo 78
4.12.3.5. NetGroupSetInfo 79
4.12.3.6. NetGroupDel 80
4.12.3.7. NetGroupDelUser 80
4.12.3.8. NetGroupGetUsers 81
4.12.3.9. NetGroupSetUsers 82
4.12.4. Local Group APIs 83
4.12.4.1. NetLocalGroupAdd 84
4.12.4.2. NetLocalGroupAddMember 85
4.12.4.3. NetLocalGroupEnum 86
4.12.4.4. NetLocalGroupGetInfo 87
4.12.4.5. NetLocalGroupSetInfo 87
4.12.4.6. NetLocalGroupDel 88
4.12.4.7. NetLocalGroupDelMember 89
4.12.4.8. NetLocalGroupGetMembers 89
4.12.4.9. NetLocalGroupSetMembers 90
4.12.5. Access APIs 91
4.12.5.1. NetAccessAdd 93

4.12.5.2. NetAccessEnum 94
4.12.5.3. NetAccessGetInfo 95
4.12.5.4. NetAccessSetInfo 96
4.12.5.5. NetAccessDel 97
4.12.5.6. NetAccessGetUserPerms 97
4.12.5.7. NetAccessCheck 98
4.13. Domain APIs 99
4.13.1. NetGetDCName 99
4.14. Transport APIs 100
4.14.1. NetServerTransportAdd 101
4.14.2. NetServerTransportDel 101
4.14.3. NetServerTransportEnum 102
4.14.4. NetWkstaTransportAdd 103
4.14.5. NetWkstaTransportDel 103
4.14.6. NetWkstaTransportEnum 104
4.15. Alert APIs 105
4.15.1. NetAlertRaise 106
4.15.2. NetAlertRaiseEx 107
4.16. Error Logging APIs 107
4.16.1. NetErrorLogClear 109
4.16.2. NetErrorLogRead 109
4.17. Configuration APIs 111
4.17.1. NetConfigGet 111
4.17.2. NetConfigGetAll 112
4.17.3. NetConfigSet 112
4.18. Statistics APIs 113
4.18.1. NetStatisticsGet 116
4.19. Auditing APIs 117
Variable-Length Data 119
4.19.1. NetAuditClear 123
4.19.2. NetAuditRead 124
4.19.3. NetAuditWrite 126
4.20. Replicator APIs 127
4.20.1. Replicator Configuration APIs 127
4.20.1.1. NetReplGetInfo 129
4.20.2. NetReplSetInfo 130
4.20.3. Replicator Export Directory APIs 130
4.20.3.1. NetReplExportDirAdd 132
4.20.4. NetReplExportDirDel 132
4.20.5. NetReplExportDirEnum 133
4.20.6. NetReplExportDirGetInfo 134
4.20.7. NetReplExportDirSetInfo 135
4.20.7.1. NetReplExportDirLock 135
4.20.7.2. NetReplExportDirUnlock 136
4.20.8. Replicator Import Directory APIs 137
4.20.8.1. NetReplImportDirAdd 138
4.20.9. NetReplImportDirDel 139
4.20.10. NetReplImportDirEnum 139
4.20.11. NetReplImportDirGetInfo 140
4.20.11.1. NetReplImportDirLock 141
4.20.11.2. NetReplImportDirUnlock 142

5. Lanman APIs That Are Not Supported In Windows NT 142
5.1. APIs with no 32 bit Net equivalents 143

5.2. APIs that only have support for remoting to downlevel 143

6. Interoperabilty Considerations 143
6.1. Requests From 16-bit LANMan Clients 143
6.2. Calling 16-bit LANMan Servers 144

7. Change History 146

1. Overview

For OS/2 based servers the LANMan APIs provided much of the functionality required for a network operating system which was missing from the local operating system. The Windows NT operating system will provide this missing set of functionality in the base. However, in order to make the migration of existing 16-bit applications as easy as possible, it is necessary to still define 32-bit equivalents of some of the LANMan APIs.

The Windows Networking APIs specified in this document are a private set of APIs designed to provide some of the API functionality that was available in LANManager 2.x. They are not the public Windows NT networking APIs. Windows NT takes some of the functionality that was previously supplied by the networking software, and moves this into the base APIs (e.g. error and audit logging, printing). Windows NT also provides a network independent set of network APIs (the WNet APIs) that allow network api's to work across different network vendors' products. If a base API or WNet API exists that could be used by your application, you should convert from the Windows networking API to the public Windows NT equivalent. There are at least three reasons to make the change now:

1.1. The WNet APIs are network independent, while the Windows networking APIs only work on LANManager networks.

1.2. The Windows networking APIs specified in this document may not be supported in future releases on Windows NT. They are provided as a interim set of APIs to assist in the porting of LANManager applications.

1.3. The Windows Networking APIs, as they are not part of the public Win32 API set, have not received the rigorous testing that the Win32 APIs have. They have been tested only by their usage internally.

In this specification, equivalent Win32 APIs will be listed that could be used in place of the Windows networking API. If at all possible, this is the API that your application should use.

This spec assumes that the reader is familiar with the existing set of 16-bit LANMan APIs. The 32-bit widening of the API data structures is specified here but the description of the functionality for each field is not covered unless the functionality has changed from that specified in the LANMAN 2.x Programmer's Reference.

2. API buffers

The RPC runtimes will allocate the buffers required to remote the APIs. This is a requirement for both efficiency and interoperability. Using the RPC runtimes to allocate the API transmission buffers results in the two significant differences between a LANMan 2.x API and windows networking API.

- For a set type API (data to the server) the API caller specifies a buffer containing the info structure relevant to the API level but does not specify the buffer length.

- For a get type API (data returned from the server) the caller does not pre-allocate a buffer for the return information. The caller passes a LPBYTE * to the API on input. On successful return the buffer pointer will contain a pointer to a buffer containing the return information. When the caller has finished processing the returned information NetApiBufferFree must be called. This simplifies the calling code as the caller does not need to guess at the size of the buffer required and will

not need to resize and reissue the API as was the case with the LANMan 2.x APIs.

2.1. API Data Alignment

All data structures specified for the Windows networking APIs will be 32-bit word aligned. The base size for an API structure element will be a DWORD.

2.2. Enumeration Buffer Lengths

Enumeration APIs will take an advisory maximum data length parameter, prefmaxlen, which allows a control on the number of bytes returned from an enumeration call. The prefered length is specified in units of 8-bit bytes. The actual API may return more than the prefered maximum length.

A Windows networking API ennumeration call will not return partial entries.

2.3. Parameter Error Reporting

Add and SetInfo APIs will return an index for a parameter in error. The caller may pass a NULL pointer for the parm_err parameter indicating that the field should not be set by the API. For remoted APIs to downlevel servers this field will be returned as PARM_ERROR_UNKNOWN by the RpcXlate conversion layer.

2.4. Parmnum For SetInfo

In order to reduce the number of RPC interfaces we need for each SetInfo API, there will be no Parmnum parameter for the 32-bit SetInfo APIs. Instead, the fields that can be set in the information structures will be set using additional infolevels. Although this will increase the number of infolevels for a given API, it will be easier to program to them. In an attempt to keep the mapping to and from downlevel calls simple, the new infolevels will be easily mapped to the old parmnum values (by subtracting a specific number i.e. PARMNUM_BASE_INFOLEVEL (whose value will be 1000)).

2.5. Parmnum For GetInfo

There will be no addition of a parmnum field for the GetInfo API.

2.6. Embedded Strings

Windows networking API info structures will not contain embedded strings. This improves the alignability of the info structures and allows for OEM flexibility in the core APIs. This does not change the feel of the LANMan 2.x APIs which support string pointers. Code porting merely requires a pointer assignment rather than a string copy.

Any API information field that is returned in an enumeration call that can be subsequently used as a key for a GetInfo call is guaranteed to be present in the enumeration buffer. If the variable length information string that would specify the key field value will not fit then the entire fixed length structure for the entry is not returned. Other variable-length fields will be returned as a NULL pointer for the case where the string will not fit as with LANMan 2.x.

2.7. Enumeration Resume Handles

Enumeration resume handles will be identifiers for the actual resume key contained in the instance data for the API. This is required for security, interoperabilty and to simplify the caller code for the API.

If a NULL is passed for the pointer to the resume handle, then no handle is stored and the ennumeration search cannot be continued. This is useful in cases where the application does not want to ennumerate all the items.

If an error is returned from an enumeration call, the resume handle must be treated as invalid and not used for any subsequent enumeration calls, i.e. the enumeration should be restarted from the beginning.

2.8. User-specific information

The LANMan 2.x APIs assumed that there would only be one user per machine. Therefore, some APIs have that assumption built in, e.g. the NetSession APIs. Since this assumption is not valid on NT, the APIs have had to be modified to take a user name as an additional parameter in order to be able to uniquely identify the information being queried or set.

2.9. Enumeration APIs

The enumeration APIs all return the number of entries read, the total entries that could have been returned if the buffer was big enough, a buffer with the entries, and a return code. The following table details what an application can expect for the various parameters (M is less than N):

Condition	entriesread	totalentries	buffer	returncode
No entries	0	0	NULL	SUCCESS
>= 1 entry	N	N	NOT NULL	SUCCESS
>= 1 entry	M	N	NOT NULL	ERROR_MORE_DATA
Failure	?	?	?	Error Code

The caller must first examine the returned status before proceeding to examine any of the other parameters for data.

Note that "totalentries" is the total number of entries that would have been returned if the return buffer was big enough - i.e. the total number of entries that were available for entries, using the current resume handle, if any, when the API was called.

2.10. SetInfo APIs

All the SetInfo APIs take a structure that contains the new information to be set. If any of the pointer fields in the structure are NULL pointers, then the corresponding field is not changed by the call. This is consistent with LANMan 2.x functionality.

In addition, the infolevels that allow GUEST access to query information do not allow callers with GUEST access to set the fields, i.e. a caller with GUEST that calls the SetInfo API will probably get an error back from the SetInfo call.

3. Windows Networking APIs

3.1. API Status

Windows networking APIs will use the LANMan 2.x error reporting convention, i.e. a successful API call returns 0, a non-zero return code indicates an error in the range of the LANMan 2.x API/OS2 errors.

An extended error range will be added for NT specific errors.

Since the Windows networking APIs will use RPC the API error definition will also be extended to include RPC error codes.

3.2. RPC Buffer Allocation Errors

Since the RPC runtime will allocate memory for send and receive buffers the API should expect to see RPC allocation errors. In the event of an RPC allocation error a resumable API handle is invalidated. This is a requirement since resumable APIs are not rewindable.

3.3. Obsolete Information Fields

Many of the information fields in the core API information structures will be obsolescent in the NT service implementation. These fields will remain in the information structure for level compatibility and will be returned with an intelligent default on NT systems.

3.4. NLS Support

There are only UNICODE versions of the Windows networking APIs. If you use these APIs you MUST define UNICODE in your source file prior to including any of the LM include files.

4. Windows Networking API Definitions.

The WIndows Networking APIs are 'stdcall' calling convention functions which return a DWORD API status;

```
#define NET_API_STATUS DWORD

#define NET_API_FUNCTION  __stdcall
```

4.1. Buffer Manipulation APIs.

For remotable APIs which return information to the caller the RPC runtime will allocate the buffer containg the return information. When the caller has completed processing the returned information NetApiBufferFree must be called. Additional buffer manipulation functions are also supported.

4.1.1. NetApiBufferAllocate

NetApiBufferAllocate allocates ByteCount bytes of memory from the heap.

NET_API_STATUS NET_API_FUNCTION
NetApiBufferAllocate (
IN DWORD *ByteCount,*
IN LPBYTE * *buffer*
);

Parameters:

ByteCount __The number of bytes to allocate.
buffer ___A pointer to the location to store the pointer to the allocated buffer.

4.1.2. NetApiBufferFree

NetApiBufferFree frees the memory allocated by NetApiBufferAllocate.

NET_API_STATUS NET_API_FUNCTION
NetApiBufferFree (IN LPVOID *buffer*
);
Parameters:

buffer ___A pointer to an API information buffer previously returned on an API call.

4.1.3. NetApiBufferReallocate

NetApiBufferReallocate changes the size of a buffer allocated with NetApiBufferAllocate.

NET_API_STATUS NET_API_FUNCTION
NetApiBufferReallocate (
IN LPVOID *OldBuffer***,**
IN DWORD *NewByteCount***,**
IN LPVOID *buffer*
);
Parameters:

OldBuffer __A pointer to the buffer to reallocate.
NewByteCount __The new size of the buffer.
buffer ___A pointer to an API information buffer previously returned on an API call.

4.1.4. NetApiBufferSize

NetApiBufferSize returns the size in bytes of the buffer allocated via NetApiBufferAllocate.

NET_API_STATUS NET_API_FUNCTION
NetApiBufferSize (
IN LPVOID *buffer,*
OUT DWORD *ByteCount*
);

Parameters:

buffer ___A pointer to an API information buffer previously returned on an API call.
ByteCount __The size of the buffer.

4.2. Service APIs.

A complete set of Service APIs are provided in the Win32 API set. These should be used in place of the NetService APIs, unless you have a requirement to control services on a downlevel server. See the Services overview in WinHelp for more details.

Service API functions control services. A service is an application that an administrator can control using the Service Controller interfaces.

Services allow administrators to control applications on the network and maintain the integrity of users' data. On a typical network, applications are shared by many users. If an administrator terminates an application running on a server, a user who has not finished working with that application can lose important data. When an application is implemented as a service, the service controller checks the status before changing the state of the service. NT Networking provides several standard services, such as the Workstation, Server, and Messenger services.

A service can be started using the Service API functions. At startup time, the service defines whether it can be stopped, paused, and continued.

NetServiceControl controls the operations of network services, and it can provide time hints to controlling applications. **NetServiceEnum** retrieves information about all started services. **NetServiceGetInfo** retrieves information about a particular started service. **NetServiceInstall** starts a network service. **NetServiceStatus** sets status and code information for a network service.

The service APIs provide service information at three levels:

```
typedef struct _SERVICE_INFO_0 {
        LPWSTR          svci0_name;     /* service name */
} SERVICE_INFO_0, *PSERVICE_INFO_0, *LPSERVICE_INFO_0;

typedef struct _SERVICE_INFO_1 {
        LPWSTR          svci1_name;
        DWORD           svci1_status;
        DWORD           svci1_code;
        DWORD           svci1_pid;
} SERVICE_INFO_1, *PSERVICE_INFO_1, *LPSERVICE_INFO_1;
```

```
typedef struct _SERVICE_INFO_2 {
        LPWSTR          svci2_name;
        DWORD           svci2_status;
        DWORD           svci2_code;
        DWORD           svci2_pid;
        LPWSTR          svci2_text;
        DWORD           svci2_specific_error;
} SERVICE_INFO_2, *PSERVICE_INFO_2, *LPSERVICE_INFO_2;
```

The service APIs are:

4.2.1. NetServiceControl

NetServiceControl controls the operations of network services.

Privilege Level

Admin privilege. Power User or Server Operator privilege is required to successfully execute **NetServiceControl** unless the opcode is SERVICE_CTRL_INTERROGATE. In this case, no special privilege is required.

```
NET_API_STATUS NET_API_FUNCTION
NetServiceControl (
        IN LPWSTR servername OPTIONAL,
        IN LPWSTR service,
        IN DWORD opcode,
        IN DWORD arg,
        OUT LPBYTE * bufptr
        );
```

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

service ___A pointer to an UNICODE string containing the name of the service to receive the control command.

opcode ___Control code to send to service

SERVICE_CTRL_INTERROGATE	Obtain the status of the service
SERVICE_CTRL_PAUSE	Pause the service
SERVICE_CTRL_CONTINUE	Continue the paused service
SERVICE_CTRL_UNINSTALL	Stop the service

arg ___Service specific contol argument to send to service

bufptr ___A pointer to the return information structure is returned in the address pointed to by bufptr.

4.2.2. NetServiceEnum

NetServiceEnum retrieves information about all started services, including paused services.

Privilege Level

No special privilege level is required to successfully execute **NetServiceEnum**.

NET_API_STATUS NET_API_FUNCTION
NetServiceEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required, 0, 1 and 2 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing service search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.2.3. NetServiceGetInfo

NetServiceGetInfo retrieves information about a particular started service.

Privilege Level

No special privilege level is required to successfully execute **NetServiceGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetServiceGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *service*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

service ___A pointer to an UNICODE string containing the name of the service.

level ___Level of information required. 0, 1 and 2 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by *bufptr*.

4.2.4. NetServiceInstall

NetServiceInstall starts a network service.

Privilege Level

Admin privilege. Power User or Server Operator privilege is required to successfully execute **NetServiceInstall**.

NET_API_STATUS NET_API_FUNCTION
NetServiceInstall (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *service*,
IN DWORD *argc*,
IN LPWSTR *argv[]*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

service ___A pointer to an UNICODE string containing the name of the service to start.

bufptr ___On return a pointer to a SERVICE_INFO_2 structure is returned in the address pointed to by *bufptr*.

cmdargs ___A pointer to a set of command line arguments to pass to the starting service as command line arguments.

4.3. Server APIs

Server API functions perform administrative tasks on a local or remote server. Any user or application with admin privilege on a local or remote server can perform administrative tasks on that server to control its operation, user access, and resource sharing. The low-level parameters that affect a server's operation can be examined and modified by calling **NetServerGetInfo** and **NetServerSetInfo**.

The LANMan 2.x server API included several fields which logically belong to other LANMan services and core NT components. For this reason the server information levels available in LANMan 2.x are no longer available in Windows networking APIs. The server specific information is available in 3 levels starting at base level 100.

The server APIs information structures are:

System Information -

Level 100 available with guest access -

```
typedef struct _SERVER_INFO_100 {
        DWORD                   sv100_platform_id;
        LPWSTR                  sv100_name;
} SERVER_INFO_100, *PSERVER_INFO_100, *LPSERVER_INFO_100;
```

Level 101 available with User access -

```
typedef struct _SERVER_INFO_101 {
        DWORD                   sv101_platform_id;
        LPWSTR                  sv101_name;
        DWORD                   sv101_version_major;
        DWORD                   sv101_version_minor;
        DWORD                   sv101_type;
        LPWSTR                  sv101_comment;
} SERVER_INFO_101, *PSERVER_INFO_101, *LPSERVER_INFO_101;
```

Level 102 available with Admin access -

```
typedef struct SERVER_INFO_102 {
        DWORD                   sv102_platform_id;
        LPWSTR                  sv102_name;
        DWORD                   sv102_version_major;
        DWORD                   sv102_version_minor;
        DWORD                   sv102_type;
        LPWSTR                  sv102_comment;
        DWORD                   sv102_users;
        DWORD                   sv102_disc;
        BOOL                    sv102_hidden;
        DWORD                   sv102_announce;
        DWORD                   sv102_anndelta;
        LPWSTR                  sv102_userpath;
} SERVER_INFO_102, *PSERVER_INFO_102, *LPSERVER_INFO_102;
```

Level 400-499 OS/2 Server information.

Level 402 available with Admin access -

```
typedef struct _SERVER_INFO_402 {
        DWORD                   sv402_ulist_mtime;
        DWORD                   sv402_glist_mtime;
        DWORD                   sv402_alist_mtime;
        LPWSTR                  sv402_alerts;
        DWORD                   sv402_security;
        DWORD                   sv402_numadmin;
        DWORD                   sv402_lanmask;
        LPWSTR                  sv402_guestacct;
        DWORD                   sv402_chdevs;
        DWORD                   sv402_chdevq;
        DWORD                   sv402_chdevjobs;
        DWORD                   sv402_connections;
        DWORD                   sv402_shares;
        DWORD                   sv402_openfiles;
        DWORD                   sv402_sessopens;
        DWORD                   sv402_sessvcs;
        DWORD                   sv402_sessreqs;
        DWORD                   sv402_opensearch;
        DWORD                   sv402_activelocks;
        DWORD                   sv402_numreqbuf;
        DWORD                   sv402_sizreqbuf;
        DWORD                   sv402_numbigbuf;
        DWORD                   sv402_numfiletasks;
        DWORD                   sv402_alertsched;
        DWORD                   sv402_erroralert;
        DWORD                   sv402_logonalert;
        DWORD                   sv402_accessalert;
        DWORD                   sv402_diskalert;
        DWORD                   sv402_netioalert;
        DWORD                   sv402_maxauditsz;
        LPWSTR                  sv402_srvheuristics;
} SERVER_INFO_402, *PSERVER_INFO_402, *LPSERVER_INFO_402;
```

Level 403 available with Admin access -

```
typedef struct _SERVER_INFO_403 {
        DWORD                   sv403_ulist_mtime;
        DWORD                   sv403_glist_mtime;
        DWORD                   sv403_alist_mtime;
        LPWSTR                  sv403_alerts;
        DWORD                   sv403_security;
        DWORD                   sv403_numadmin;
        DWORD                   sv403_lanmask;
        LPWSTR                  sv403_guestacct;
        DWORD                   sv403_chdevs;
        DWORD                   sv403_chdevq;
        DWORD                   sv403_chdevjobs;
        DWORD                   sv403_connections;
        DWORD                   sv403_shares;
        DWORD                   sv403_openfiles;
        DWORD                   sv403_sessopens;
        DWORD                   sv403_sessvcs;
        DWORD                   sv403_sessreqs;
        DWORD                   sv403_opensearch;
        DWORD                   sv403_activelocks;
        DWORD                   sv403_numreqbuf;
        DWORD                   sv403_sizreqbuf;
        DWORD                   sv403_numbigbuf;
        DWORD                   sv403_numfiletasks;
        DWORD                   sv403_alertsched;
        DWORD                   sv403_erroralert;
        DWORD                   sv403_logonalert;
        DWORD                   sv403_accessalert;
        DWORD                   sv403_diskalert;
        DWORD                   sv403_netioalert;
        DWORD                   sv403_maxauditsz;
        LPWSTR                  sv403_srvheuristics;
        DWORD                   sv403_auditedevents;
        DWORD                   sv403_autoprofile;
        LPWSTR                  sv403_autopath;
} SERVER_INFO_403, *PSERVER_INFO_403, *LPSERVER_INFO_403;
```

Level 500-599 NT Server information.

Level 502 available with Admin access -

```
typedef struct _SERVER_INFO_502 {
        DWORD                   sv502_sessopens;
        DWORD                   sv502_sessvcs;
        DWORD                   sv502_opensearch;
        DWORD                   sv502_sizreqbuf;
        DWORD                   sv502_initworkitems;
        DWORD                   sv502_maxworkitems;
        DWORD                   sv502_rawworkitems;
        DWORD                   sv502_irpstacksize;
        DWORD                   sv502_maxrawbuflen;
        DWORD                   sv502_sessusers;
        DWORD                   sv502_sessconns;
        DWORD                   sv502_maxpagedmemoryusage;
```

```
        DWORD                   sv502_maxnonpagedmemoryusage;
        BOOL                    sv502_enablesoftcompat;
        BOOL                    sv502_enableforcedlogoff;
        BOOL                    sv502_timesource;
        BOOL                    sv502_acceptdownlevelapis;
        BOOL                    sv502_lmannounce;
} SERVER_INFO_502, *PSERVER_INFO_502, *LPSERVER_INFO_502;
typedef struct _SERVER_INFO_503 {
        DWORD                   sv503_sessopens;
        DWORD                   sv503_sessvcs;
        DWORD                   sv503_opensearch;
        DWORD                   sv503_sizreqbuf;
        DWORD                   sv503_initworkitems;
        DWORD                   sv503_maxworkitems;
        DWORD                   sv503_rawworkitems;
        DWORD                   sv503_irpstacksize;
        DWORD                   sv503_maxrawbuflen;
        DWORD                   sv503_sessusers;
        DWORD                   sv503_sessconns;
        DWORD                   sv503_maxpagedmemoryusage;
        DWORD                   sv503_maxnonpagedmemoryusage;
        BOOL                    sv503_enablesoftcompat;
        BOOL                    sv503_enableforcedlogoff;
        BOOL                    sv503_timesource;
        BOOL                    sv503_acceptdownlevelapis;
        BOOL                    sv503_lmannounce;
        LPTSTR                  sv503_domain;
        DWORD                   sv503_maxcopyreadlen;
        DWORD                   sv503_maxcopywritelen;
        DWORD                   sv503_minkeepsearch;
        DWORD                   sv503_maxkeepsearch;
        DWORD                   sv503_minkeepcomplsearch;
        DWORD                   sv503_maxkeepcomplsearch;
        DWORD                   sv503_threadcountadd;
        DWORD                   sv503_numblockthreads;
        DWORD                   sv503_scavtimeout;
        DWORD                   sv503_minrcvqueue;
        DWORD                   sv503_minfreeworkitems;
        DWORD                   sv503_xactmemsize;
        DWORD                   sv503_threadpriority;
        DWORD                   sv503_maxmpxct;
        DWORD                   sv503_oplockbreakwait;
        DWORD                   sv503_oplockbreakresponsewait;
        BOOL                    sv503_enableoplocks;
        BOOL                    sv503_enableoplockforceclose;
        BOOL                    sv503_enablefcbopens;
        BOOL                    sv503_enableraw;
        BOOL                    sv503_enabledpc;
        BOOL                    sv503_enablemdlio;
        BOOL                    sv503_enablefastio;
} SERVER_INFO_503, *PSERVER_INFO_503, *LPSERVER_INFO_503;
```

The following infolevels are only valid for **NetServerSetInfo** and replace the older way of passing in a *Parmnum* to set a specific field.

The following are supported on downlevel systems (i.e. LANMan 2.x) as well:

```
typedef struct _SERVER_INFO_1005 {
        DWORD                   sv1005_comment;
} SERVER_INFO_1005, *PSERVER_INFO_1005, *LPSERVER_INFO_1005;

typedef struct _SERVER_INFO_1107 {
        DWORD                   sv1107_users;
} SERVER_INFO_1107, *PSERVER_INFO_1107, *LPSERVER_INFO_1107;

typedef struct _SERVER_INFO_1010 {
        DWORD                   sv1010_disc;
} SERVER_INFO_1010, *PSERVER_INFO_1010, *LPSERVER_INFO_1010;

typedef struct _SERVER_INFO_1016 {
        BOOL                    sv1016_hidden;
} SERVER_INFO_1016, *PSERVER_INFO_1016, *LPSERVER_INFO_1016;

typedef struct _SERVER_INFO_1017 {
        DWORD                   sv1017_announce;
} SERVER_INFO_1017, *PSERVER_INFO_1017, *LPSERVER_INFO_1017;

typedef struct _SERVER_INFO_1018 {
        DWORD                   sv1018_anndelta;
} SERVER_INFO_1018, *PSERVER_INFO_1018, *LPSERVER_INFO_1018;

typedef struct _SERVER_INFO_1037 {
        DWORD                   sv1037_alertsched;
} SERVER_INFO_1037, *PSERVER_INFO_1037, *LPSERVER_INFO_1037;

typedef struct _SERVER_INFO_1038 {
        DWORD                   sv1038_erroralert;
} SERVER_INFO_1038, *PSERVER_INFO_1038, *LPSERVER_INFO_1038;

typedef struct _SERVER_INFO_1039 {
        DWORD                   sv1039_logonalert;
} SERVER_INFO_1039, *PSERVER_INFO_1039, *LPSERVER_INFO_1039;

typedef struct _SERVER_INFO_1040 {
        DWORD                   sv1040_accessalert;
} SERVER_INFO_1040, *PSERVER_INFO_1040, *LPSERVER_INFO_1040;

typedef struct _SERVER_INFO_1041 {
        DWORD                   sv1041_diskalert;
} SERVER_INFO_1041, *PSERVER_INFO_1041, *LPSERVER_INFO_1041;

typedef struct _SERVER_INFO_1042 {
        DWORD                   sv1042_netioalert;
} SERVER_INFO_1042, *PSERVER_INFO_1042, *LPSERVER_INFO_1042;

typedef struct _SERVER_INFO_1043 {
        DWORD                   sv1043_maxauditsz;
} SERVER_INFO_1043, *PSERVER_INFO_1043, *LPSERVER_INFO_1043;
```

The following do not do anything on downlevel systems (i.e. LANMan 2.x) since they cannot be modified on the fly on those systems:

```
typedef struct _SERVER_INFO_1009 {
        DWORD                   sv1009_users;
} SERVER_INFO_1009, *PSERVER_INFO_1009, *LPSERVER_INFO_1009;

typedef struct _SERVER_INFO_1021 {
        LPWSTR                  sv1021_userpath;
} SERVER_INFO_1021, *PSERVER_INFO_1021, *LPSERVER_INFO_1021;

typedef struct _SERVER_INFO_1022 {
        DWORD                   sv1022_chdevs;
} SERVER_INFO_1022, *PSERVER_INFO_1022, *LPSERVER_INFO_1022;

typedef struct _SERVER_INFO_1028 {
        DWORD                   sv1028_sessopens;
} SERVER_INFO_1028, *PSERVER_INFO_1028, *LPSERVER_INFO_1028;

typedef struct _SERVER_INFO_1029 {
        DWORD                   sv1029_opensearch;
} SERVER_INFO_1029, *PSERVER_INFO_1029, *LPSERVER_INFO_1029;
```

The following are not supported on downlevel systems (i.e. LANMan 2.x) since these are NT-only fields:

```
typedef struct _SERVER_INFO_1001 {
        DWORD                   sv1001_maxworkitems;
} SERVER_INFO_1001, *PSERVER_INFO_1001, *LPSERVER_INFO_1001;

typedef struct _SERVER_INFO_1002 {
        DWORD                   sv1002_maxrawbuflen;
} SERVER_INFO_1002, *PSERVER_INFO_1002, *LPSERVER_INFO_1002;

typedef struct _SERVER_INFO_1003 {
        DWORD                   sv1003_sessusers;
} SERVER_INFO_1003, *PSERVER_INFO_1003, *LPSERVER_INFO_1003;

typedef struct _SERVER_INFO_1004 {
        DWORD                   sv1004_sesscons;
} SERVER_INFO_1004, *PSERVER_INFO_1004, *LPSERVER_INFO_1004;

typedef struct _SERVER_INFO_1006 {
        DWORD                   sv1006_enablesoftcompat;
} SERVER_INFO_1006, *PSERVER_INFO_1006, *LPSERVER_INFO_1006;
```

For **NetServerSetInfo**, parmnum values refer to the fields in the _SERVER_INFO structures as follows. These values are used when indicating an error in a specific parameter via *parm_err*.

parmnum value	Field in server_info struct
SV_NAME_PARMNUM	sv_name
SV_VERSION_MAJOR_PARMNUM	sv_version_major

SV_VERSION_MINOR_PARMNUM	sv_version_minor
SV_TYPE_PARMNUM	sv_type
SV_COMMENT_PARMNUM	sv_comment
SV_USERS_PARMNUM	sv_users
SV_DISC_PARMNUM	sv_disc
SV_HIDDEN_PARMNUM	sv_hidden
SV_ANNOUNCE_PARMNUM	sv_announce
SV_ANNDELTA_PARMNUM	sv_anndelta
SV_USERPATH_PARMNUM	sv_userpath
SV_ULIST_MTIME_PARMNUM	sv_ulist_mtime
SV_GLIST_MTIME_PARMNUM	sv_glist_mtime
SV_ALIST_MTIME_PARMNUM	sv_alist_mtime
SV_ALERTS_PARMNUM	sv_alerts
SV_SECURITY_PARMNUM	sv_security
SV_NUMADMIN_PARMNUM	sv_numadmin
SV_LANMASK_PARMNUM	sv_lanmask
SV_GUESTACC_PARMNUM	sv_guestacc
SV_CHDEVS_PARMNUM	sv_chdevs
SV_CHDEVQ_PARMNUM	sv_chdevq
SV_CHDEVJOBS_PARMNUM	sv_chdevjobs
SV_CONNECTIONS_PARMNUM	sv_connections
SV_SHARES_PARMNUM	sv_shares
SV_OPENFILES_PARMNUM	sv_openfiles
SV_SESSOPENS_PARMNUM	sv_sessopens
SV_SESSVCS_PARMNUM	sv_sessvcs
SV_SESSREQS_PARMNUM	sv_sessreqs
SV_OPENSEARCH_PARMNUM	sv_opensearch
SV_ACTIVELOCKS_PARMNUM	sv_activelocks
SV_NUMREQBUF_PARMNUM	sv_numreqbuf
SV_SIZREQBUF_PARMNUM	sv_sizreqbuf
SV_NUMBIGBUF_PARMNUM	sv_numbigbuf
SV_NUMFILETASKS_PARMNUM	sv_numfiletasks
SV_ALERTSCHED_PARMNUM	sv_alertsched
SV_ERRORALERT_PARMNUM	sv_erroralert
SV_LOGONALERT_PARMNUM	sv_logonalert
SV_ACCESSALERT_PARMNUM	sv_accessalert
SV_DISKALERT_PARMNUM	sv_diskalert
SV_NETIOALERT_PARMNUM	sv_netioalert
SV_MAXAUDITSZ_PARMNUM	sv_maxauditsz
SV_SRVHEURISTICS_PARMNUM	sv_srvheuristics
SV_AUDITEDEVENTS_PARMNUM	sv_auditedevents
SV_AUTOPROFILE_PARMNUM	sv_autoprofile
SV_MAXWORKITEMS_PARMNUM	sv_maxworkitems
SV_RAWWORKITEMS_PARMNUM	sv_rawworkitems
SV_IRPSTACKSIZE_PARMNUM	sv_irpstacksize
SV_SESSUSERS_PARMNUM	sv_sessusers
SV_SESSCONNS_PARMNUM	sv_sessconns
SV_MAXNONPAGEDMEMORYUSAGE_PARMNUM	sv_maxnonpagedmemoryusage
SV_MAXPAGEDMEMORYUSAGE_PARMNUM	sv_maxpagedmmeoryusage
SV_ENABLEOFTCOMPAT_PARMNUM	sv_enablesoftcompat

The server APIs are:

4.3.1. NetServerEnum

Use the WNetEnumResource Win32 API if the information you require is returned by it. The WNetEnumResource will return the type (disk, printer, ...), name and comment for a server.

NetServerEnum lists all servers of the specified type(s) that are visible in the specified domain(s). For example, an application can call **NetServerEnum** to list all domain controllers only or all SQL servers only.

Privilege Level

No special privilege level is required to successfully execute **NetServerEnum**.

NET_API_STATUS NET_API_FUNCTION
NetServerEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN DWORD *servertype*,
IN LPWSTR *domain*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. 100 and 101 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

servertype __A DWORD mask which filters server entries to return from the enumeration. The defines mask bits are:

SV_TYPE_WORKSTATION	0x00000001	All LAN Manager workstation
SV_TYPE_SERVER	0x00000002	All LAN Manager server
SV_TYPE_SQLSERVER	0x00000004	Any server running with SQL server
SV_TYPE_DOMAIN_CTRL	0x00000008	Primary domain controller
SV_TYPE_DOMAIN_BAKCTRL	0x00000010	Backup domain controller
SV_TYPE_TIMESOURCE	0x00000020	Server running the timesource service
SV_TYPE_AFP	0x00000040	Apple File Protocol servers
SV_TYPE_NOVELL	0x00000080	Novell servers
SV_TYPE_DOMAIN_MEMBER	0x00000100	Domain Member
SV_TYPE_PRINT	0x00000200	Server sharing print queue
SV_TYPE_DIALIN	0x00000400	Server running dialin service.
SV_TYPE_XENIX_SERVER	0x00000800	Xenix server
SV_TYPE_NT	0x00001000	NT server
SV_TYPE_WFW	0x00002000	Server running Windows for Workgroups
SV_TYPE_POTENTIAL_BROWSER	0x00010000	Server that can run the browser service
SV_TYPE_BACKUP_BROWSER	0x00020000	Server running a browser service as backup
SV_TYPE_MASTER_BROWSER	0x00040000	Server running the master browser service
SV_TYPE_DOMAIN_MASTER	0x00080000	Server running the domain master browser
SV_TYPE_ALL	0xffffffff	All servers

domain __A pointer to an UNICODE string containing the name of the domain for which a list of servers is to returned. The domain must be the primary domain, one of the other domains for the workstation or the logon domain.

resumehandle __On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing server search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.3.2. NetServerGetInfo

NetServerGetInfo retrieves information about the specified server.

Privilege Level

Admin privilege or accounts, comm, print, or server operator privilege is required to successfully execute **NetServerGetInfo** at level 102 or higher. No special privilege is required for level 100 or level 101 calls.

NET_API_STATUS NET_API_FUNCTION
NetServerGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
);

Parameters:

servername __A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level __Level of information required. 100, 101 and 102 are valid for all platforms. 302, 402, 403, 502 are valid for the appropriate platform.

bufptr __On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.3.3. NetServerSetInfo

NetServerSetInfo sets a server's operating parameters; it can set them individually or collectively. This information is stored such that it remains in effect after the system has been reinitialized.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetServerSetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetServerSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername __A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level __Level of information to set. 100, 101 and 102 are valid for all platforms. 302, 402, 403, 502 are valid for the appropriate platform. In addition, 1001 - 1006, 1009 - 1011, 1016 - 1018, 1021, 1022, 1028, 1029, 1037 - 1043 are valid based on the restrictions for downlevel systems described above.

buf __A pointer to a buffer containing the server information.

parm_err __Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.3.4. NetServerDiskEnum

NetServerDiskEnum retrieves a list of disk drives on a server.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetServerDiskEnum** on a remote computer. No special privilege is required for local calls.

NET_API_STATUS NET_API_FUNCTION
NetServerDiskEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. 0 is the only valid level.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing server disk search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.4. Workstation and WorkstationUser APIs

Wksta API functions perform administrative tasks on a local or remote workstation. Any user or application with admin privilege on a local or remote server can perform administrative tasks on that workstation to control its operation, user access, and resource sharing. The low-level parameters that affect a workstation's operation can be examined and modified by calling **NetWkstaGetInfo** and **NetWkstaSetInfo**.

The workstation API data structures are restructured from those of LANMan 2.x to allow the information to be grouped by type and privilege. The LANMan 2.x workstation information format is discontinued due to the following problems:

- The base level (0 and 1) were not grouped by accessibility such that a non superset level (level 10) was required to allow guest access to the information.

- Platform specific implementation information was included in the base levels such that every platform had to return all information including a default for non-relevant fields. This grew the size of the information structures unneccessarily, making the API cumbersome to use.

The workstation APIs allow access to two discrete groups of workstation information:

- System information.
- Platform specific information (NT, OS/2, DOS, etc.)

The WkstaUser APIs allow access to user-specific information. The user-specific information is separated from the workstation information because there can be more than one user on a workstation.

Within each group the fields are categorized by security access such that the guest accessible fields are a subset of the user accessible fields which are a subset of the admin accessible fields.
The system information structure contains a platform base number which identifies the levels and format of the platform specific information structures.

The workstation APIs information structures are:

System Information -

Level 100 available with guest access -

```
typedef struct _WKSTA_INFO_100 {
        DWORD           wki100_platform_id;
        LPWSTR          wki100_computername;
        LPWSTR          wki100_langroup;
        DWORD           wki100_ver_major;
        DWORD           wki100_ver_minor;
} WKSTA_INFO_100, *PWKSTA_INFO_100, *LPWKSTA_INFO_100;
```

Level 101 available with User access -

```
typedef struct _WKSTA_INFO_101 {
        DWORD           wki101_platform_id;
        LPWSTR          wki101_computername;
        LPWSTR          wki101_langroup;
        DWORD           wki101_ver_major;
        DWORD           wki101_ver_minor;
        LPWSTR          wki101_lanroot;
} WKSTA_INFO_101, *PWKSTA_INFO_101, *LPWKSTA_INFO_101;
```

Level 102 available with Admin access -

```
typedef struct _WKSTA_INFO_102 {
        DWORD           wki102_platform_id;
        LPWSTR          wki102_computername;
        LPWSTR          wki102_langroup;
        DWORD           wki102_ver_major;
        DWORD           wki102_ver_minor;
        LPWSTR          wki102_lanroot;
        DWORD           wki102_logged_on_users;
} WKSTA_INFO_102, *PWKSTA_INFO_102, *LPWKSTA_INFO_102;
```

Platform Specific Information

Level 300-399 Dos Workstation information.

Level 300 available with guest access -

Level 302 available with Admin access -

```
typedef struct _WKSTA_INFO_302 {
        DWORD           wki302_char_wait;
        DWORD           wki302_collection_time;
        DWORD           wki302_maximum_collection_count;
        DWORD           wki302_keep_conn;
        DWORD           wki302_keep_search;
        DWORD           wki302_max_cmds;
        DWORD           wki302_num_work_buf;
        DWORD           wki302_siz_work_buf;
        DWORD           wki302_max_wrk_cache;
        DWORD           wki302_sess_timeout;
        DWORD           wki302_siz_error;
        DWORD           wki302_num_alerts;
        DWORD           wki302_num_services;
        DWORD           wki302_errlog_sz;
        DWORD           wki302_print_buf_time;
        DWORD           wki302_num_char_buf;
        DWORD           wki302_siz_char_buf;
        LPWSTR          wki302_wrk_heuristics;
        DWORD           wki302_mailslots;
        DWORD           wki302_num_dgram_buf;
} WKSTA_INFO_302, *PWKSTA_INFO_302, *LPWKSTA_INFO_302;
```

Level 400-499 OS/2 Workstation information.

Level 402 available with Admin access -

```
typedef struct _WKSTA_INFO_402 {
        DWORD           wki402_char_wait;
        DWORD           wki402_collection_time;
        DWORD           wki402_maximum_collection_count;
        DWORD           wki402_keep_conn;
        DWORD           wki402_keep_search;
        DWORD           wki402_max_cmds;
        DWORD           wki402_num_work_buf;
        DWORD           wki402_siz_work_buf;
        DWORD           wki402_max_wrk_cache;
        DWORD           wki402_sess_timeout;
        DWORD           wki402_siz_error;
        DWORD           wki402_num_alerts;
        DWORD           wki402_num_services;
        DWORD           wki402_errlog_sz;
        DWORD           wki402_print_buf_time;
        DWORD           wki402_num_char_buf;
        DWORD           wki402_siz_char_buf;
        LPWSTR          wki402_wrk_heuristics;
        DWORD           wki402_mailslots;
        DWORD           wki402_num_dgram_buf;
        DWORD           wki402_max_threads;
} WKSTA_INFO_402, *PWKSTA_INFO_402, *LPWKSTA_INFO_402;
```

Level 500-599 NT Workstation information.

Level 502 available with Admin access -

```
typedef struct _WKSTA_INFO_502 {
        DWORD           wki502_char_wait;
        DWORD           wki502_collection_time;
        DWORD           wki502_maximum_collection_count;
        DWORD           wki502_keep_conn;
        DWORD           wki502_max_cmds;
        DWORD           wki502_sess_timeout;
        DWORD           wki502_siz_char_buf;
        DWORD           wki502_max_threads;
        DWORD           wki502_lock_quota;
        DWORD           wki502_lock_increment;
        DWORD           wki502_lock_maximum;
        DWORD           wki502_pipe_increment;
        DWORD           wki502_pipe_maximum;
        DWORD           wki502_cache_file_timeout;
        DWORD           wki502_dormant_file_limit;
        DWORD           wki502_read_ahead_throughput;
        DWORD           wki502_num_mailslot_buffers;
        DWORD           wki502_num_srv_announce_buffers;
```

```
        DWORD           wki502_dgreceiver_threads;
        BOOL            wki502_use_opportunistic_locking;
        BOOL            wki502_use_unlock_behind;
        BOOL            wki502_use_close_behind;
        BOOL            wki502_buf_named_pipes;
        BOOL            wki502_use_lock_read_unlock;
        BOOL            wki502_utilize_nt_caching;
        BOOL            wki502_use_raw_read;
        BOOL            wki502_use_raw_write;
        BOOL            wki502_use_write_raw_data;
        BOOL            wki502_use_encryption;
        BOOL            wki502_buf_files_deny_write;
        BOOL            wki502_buf_read_only_files;
        BOOL            wki502_force_core_create_mode;
        BOOL            wki502_use_512_byte_max_transfer;
} WKSTA_INFO_502, *PWKSTA_INFO_502, *LPWKSTA_INFO_502;
```

This format of workstation information allows for a more logical future extension. When new information is to be added to any set of guest,user and admin information a new set of levels is created to include the current infromation plus the new information.

The following infolevels are only valid for **NetWkstaSetInfo** and replace the older way of passing in a *Parmnum* to set a specific field.

The following are supported on downlevel systems (i.e. LANMan 2.x) as well:

```
typedef struct _WKSTA_INFO_1010 {
        DWORD                   wki1010_char_wait;
} WKSTA_INFO_1010, *PWKSTA_INFO_1010, *LPWKSTA_INFO_1010;

typedef struct _WKSTA_INFO_1011 {
        DWORD                   wki1011_collection_time;
} WKSTA_INFO_1011, *PWKSTA_INFO_1011, *LPWKSTA_INFO_1011;

typedef struct _WKSTA_INFO_1012 {
        DWORD                   wki1012_maximum_collection_count;
} WKSTA_INFO_1012, *PWKSTA_INFO_1012, *LPWKSTA_INFO_1012;

typedef struct _WKSTA_INFO_1027 {
        DWORD                   wki1027_errlog_sz;
} WKSTA_INFO_1027, *PWKSTA_INFO_1027, *LPWKSTA_INFO_1027;

typedef struct _WKSTA_INFO_1028 {
        DWORD                   wki1028_print_buf_time;
} WKSTA_INFO_1028, *PWKSTA_INFO_1028, *LPWKSTA_INFO_1028;

typedef struct _WKSTA_INFO_1032 {
        DWORD                   wki1032_wrk_heuristics;
} WKSTA_INFO_1032, *PWKSTA_INFO_1032, *LPWKSTA_INFO_1032;

typedef struct _WKSTA_INFO_1035 {
        LPWSTR                  wki1035_oth_domains;
} WKSTA_INFO_1035, *PWKSTA_INFO_1035, *LPWKSTA_INFO_1035;
```

The following do nothing on downlevel systems (i.e. LANMan 2.x) since these fields cannot be set on them:

```
typedef struct _WKSTA_INFO_1013 {
        DWORD                   wki1013_keep_conn;
} WKSTA_INFO_1013, *PWKSTA_INFO_1013, *LPWKSTA_INFO_1013;

typedef struct _WKSTA_INFO_1018 {
        DWORD                   wki1018_sess_timeout;
} WKSTA_INFO_1018, *PWKSTA_INFO_1018, *LPWKSTA_INFO_1018;

typedef struct _WKSTA_INFO_1023 {
        DWORD                   wki1023_siz_char_buf;
} WKSTA_INFO_1023, *PWKSTA_INFO_1023, *LPWKSTA_INFO_1023;

typedef struct _WKSTA_INFO_1033 {
        DWORD                   wki1033_max_threads;
} WKSTA_INFO_1033, *PWKSTA_INFO_1033, *LPWKSTA_INFO_1033;
```

The following are not supported on downlevel systems (i.e. LANMan 2.x):

```
typedef struct _WKSTA_INFO_1041 {
        DWORD                   wki1041_lock_quota;
} WKSTA_INFO_1041, *PWKSTA_INFO_1041, *LPWKSTA_INFO_1041;

typedef struct _WKSTA_INFO_1042 {
        DWORD                   wki1042_lock_increment;
} WKSTA_INFO_1042, *PWKSTA_INFO_1042, *LPWKSTA_INFO_1042;

typedef struct _WKSTA_INFO_1043 {
        DWORD                   wki1043_lock_maximum;
} WKSTA_INFO_1043, *PWKSTA_INFO_1043, *LPWKSTA_INFO_1043;

typedef struct _WKSTA_INFO_1044 {
        DWORD                   wki1044_pipe_increment;
} WKSTA_INFO_1044, *PWKSTA_INFO_1044, *LPWKSTA_INFO_1044;

typedef struct _WKSTA_INFO_1045 {
        DWORD                   wki1045_pipe_maximum;
} WKSTA_INFO_1045, *PWKSTA_INFO_1045, *LPWKSTA_INFO_1045;

typedef struct _WKSTA_INFO_1046 {
        DWORD                   wki1046_dormant_file_limit;
} WKSTA_INFO_1046, *PWKSTA_INFO_1046, *LPWKSTA_INFO_1046;

typedef struct _WKSTA_INFO_1047 {
        DWORD                   wki1047_cache_file_timeout;
} WKSTA_INFO_1047, *PWKSTA_INFO_1047, *LPWKSTA_INFO_1047;

typedef struct _WKSTA_INFO_1048 {
        BOOL                    wki1048_use_opportunity_locking;
} WKSTA_INFO_1048, *PWKSTA_INFO_1048, *LPWKSTA_INFO_1048;
```

```
typedef struct _WKSTA_INFO_1049 {
        BOOL                    wki1049_use_unlock_behind;
} WKSTA_INFO_1049, *PWKSTA_INFO_1049, *LPWKSTA_INFO_1049;

typedef struct _WKSTA_INFO_1050 {
        BOOL                    wki1050_use_close_behind;
} WKSTA_INFO_1050, *PWKSTA_INFO_1050, *LPWKSTA_INFO_1050;

typedef struct _WKSTA_INFO_1051 {
        BOOL                    wki1051_buf_named_pipes;
} WKSTA_INFO_1051, *PWKSTA_INFO_1051, *LPWKSTA_INFO_1051;

typedef struct _WKSTA_INFO_1052 {
        BOOL                    wki1052_use_lock_read_unlock;
} WKSTA_INFO_1052, *PWKSTA_INFO_1052, *LPWKSTA_INFO_1052;

typedef struct _WKSTA_INFO_1053 {
        BOOL                    wki1053_utilize_nt_caching;
} WKSTA_INFO_1053, *PWKSTA_INFO_1053, *LPWKSTA_INFO_1053;

typedef struct _WKSTA_INFO_1054 {
        BOOL                    wki1054_use_raw_read;
} WKSTA_INFO_1054, *PWKSTA_INFO_1054, *LPWKSTA_INFO_1054;

typedef struct _WKSTA_INFO_1055 {
        BOOL                    wki1055_use_raw_write;
} WKSTA_INFO_1055, *PWKSTA_INFO_1055, *LPWKSTA_INFO_1055;

typedef struct _WKSTA_INFO_1056 {
        BOOL                    wki1056_use_write_raw_data;
} WKSTA_INFO_1056, *PWKSTA_INFO_1056, *LPWKSTA_INFO_1056;

typedef struct _WKSTA_INFO_1057 {
        BOOL                    wki1057_use_encryption;
} WKSTA_INFO_1057, *PWKSTA_INFO_1057, *LPWKSTA_INFO_1057;

typedef struct _WKSTA_INFO_1058 {
        BOOL                    wki1058_buf_files_deny_write;
} WKSTA_INFO_1058, *PWKSTA_INFO_1058, *LPWKSTA_INFO_1058;

typedef struct _WKSTA_INFO_1059 {
        BOOL                    wki1059_buf_read_only_files;
} WKSTA_INFO_1059, *PWKSTA_INFO_1059, *LPWKSTA_INFO_1059;

typedef struct _WKSTA_INFO_1060 {
        BOOL                    wki1060_force_core_create_mode;
} WKSTA_INFO_1060, *PWKSTA_INFO_1060, *LPWKSTA_INFO_1060;

typedef struct _WKSTA_INFO_1061 {
        BOOL                    wki1061_use_512_byte_max_transfer;
} WKSTA_INFO_1061, *PWKSTA_INFO_1061, *LPWKSTA_INFO_1061;
```

```
typedef struct _WKSTA_INFO_1062 {
        DWORD                   wki1062_read_ahead_throughput;
} WKSTA_INFO_1062, *PWKSTA_INFO_1062, *LPWKSTA_INFO_1062;
```

The WkstaUser API information structures are:

Level 0 and 1 available with User access -

```
typedef struct _WKSTA_USER_INFO_0 {
        LPWSTR          wkui0_username;
} WKSTA_USER_INFO_0, *PWKSTA_USER_INFO_0,
*LPWKSTA_USER_INFO_0;


typedef struct _WKSTA_USER_INFO_1 {
        LPWSTR          wkui1_username;
        LPWSTR          wkui1_logon_domain;
        LPWSTR          wkui1_logon_server;
        LPWSTR          wkui1_oth_domains;
} WKSTA_USER_INFO_1, *PWKSTA_USER_INFO_1,
*LPWKSTA_USER_INFO_1;

typedef struct _WKSTA_USER_INFO_1101 {
        LPTSTR          wkui1101_oth_domains;
} WKSTA_USER_INFO_1101, *PWKSTA_USER_INFO_1101,
  *LPWKSTA_USER_INFO_1101;
```

For **NetWkstaSetInfo**, parmnum values refer to the fields in the wksta_info structure as follows. These values are used when indicating an error in a specific parameter via *parm_err*.

parmnum value	Field in wksta_info struct

WKSTA_PLATFORM_ID_PARMNUM	wki_platform_id
WKSTA_COMPUTERNAME_PARMNUM	wki_computername
WKSTA_LANGROUP_PARMNUM	wki_langroup
WKSTA_OTH_DOMAINS_PARMNUM	wki_oth_domains
WKSTA_VER_MAJOR_PARMNUM	wki_ver_major
WKSTA_VER_MINOR_PARMNUM	wki_ver_minor
WKSTA_LOGGED_ON_USERS_PARMNUM	wki_logged_on_users
WKSTA_LANROOT_PARMNUM	wki_lanroot
WKSTA_LOGON_DOMAIN_PARMNUM	wki_logon_domain
WKSTA_LOGON_SERVER_PARMNUM	wki_logon_server
WKSTA_CHARWAIT_PARMNUM	wki_char_wait
WKSTA_CHARTIME_PARMNUM	wki_collection_time
WKSTA_CHARCOUNT_PARMNUM	wki_maximum_collection_count
WKSTA_KEEPCONN_PARMNUM	wki_keep_conn
WKSTA_KEEPSEARCH_PARMNUM	wki_keep_search
WKSTA_MAXCMDS_PARMNUM	wki_max_cmds
WKSTA_NUMWORKBUF_PARMNUM	wki_num_work_buf
WKSTA_MAXWRKCACHE_PARMNUM	wki_max_wrk_cache
WKSTA_SESSTIMEOUT_PARMNUM	wki_sess_timeout
WKSTA_SIZERROR_PARMNUM	wki_siz_error
WKSTA_NUMALERTS_PRAMNUM	wki_num_alerts
WKSTA_NUMSERVICES_PARMNUM	wki_num_services
WKSTA_ERRLOGSZ_PARMNUM	wki_errlog_sz
WKSTA_PRINTBUFTIME_PARMNUM	wki_print_buf_time
WKSTA_NUMCHARBUF_PARMNUM	wki_num_char_buf
WKSTA_SIZCHARBUF_PARMNUM	wki_siz_char_buf
WKSTA_WRKHEURISTICS_PARMNUM	wki_wrk_heuristics
WKSTA_MAILSLOTS_PRAMNUM	wki_mailslots
WKSTA_MAXTHREADS_PARMNUM	wki_max_threads
WKSTA_SIZWORKBUF_PARMNUM	wki_siz_work_buf
WKSTA_DORMANTTIMEOUT_PARMNUM	wki_dormant_timeout
WKSTA_LOCKQUOTA_PARMNUM	wki_lock_quota
WKSTA_LOCKINCREMENT_PARMNUM	wki_lock_increment
WKSTA_LOCKMAXIMUM_PARMNUM	wki_lock_maximum
WKSTA_PIPEINCREMENT_PRAMNUM	wki_pipe_increment
WKSTA_PIPEMAXIMUM_PARMNUM	wki_pipe_maximum
WKSTA_RAWREADTHRESHOLD_PARMNUM	wki_raw_read_threshold
WKSTA_USEOPLOCKING_PARMNUM	wki_use_opportunistic_locking
WKSTA_USEOPBATCH_PARMNUM	wki_use_op_batch
WKSTA_USEUNLOCKBEHIND_PARMNUM	wki_use_unlock_behind
WKSTA_USECLOSEBEHIND_PARMNUM	wki_use_close_behind
WKSTA_BUFNAMEDPIPES_PARMNUM	wksta_buf_named_pipes
WKSTA_USELOCKANDREADANDUNLOCK_PARMNUM	wki_use_lock_and_read_and_unlock
WKSTA_UTILIZENTCACHING_PARMNUM	wki_utilize_nt_caching
WKSTA_USERAWREAD_PARMNUM	wki_use_raw_read
WKSTA_USEWRITERAWWITHDATA_PARMNUM	wki_use_write_raw_with_data
WKSTA_USEENCRYPTION_PARMNUM	wki_use_encryption
WKSTA_BUFFILESWITHDENYWRITE_PARMNUM	wki_buf_files_with_deny_write
WKSTA_BUFREADONLYFILES_PRAMNUM	wki_buf_read_only_files
WKSTA_FORCECORECREATEMODE_PARMNUM	wki_force_core_create_mode
WKSTA_USE512BYTESMAXTRANSFER_PARMNUM	wki_use_512_bytes_max_transfer

The workstation APIs are:

4.4.1. NetWkstaGetInfo

NetWkstaGetInfo returns information about the configuration elements for a workstation.

Privilege Level

Privilege requirements are described with the data structures for the infolevels above.

NET_API_STATUS NET_API_FUNCTION
NetWkstaGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. 100, 101, 102, 302, 402 and 502 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.4.2. NetWkstaSetInfo

NetWkstaSetInfo configures a workstation. This information is stored such that it remains in effect after the system has been reinitialized.

Privilege Level

Admin privilege is required to successfully execute **NetWkstaSetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetWkstaSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information to set. 100, 101, 102, ~~201, 202,~~ 302, 402, and 502 are valid. In addition, 1010 - 1013, 1018, 1023, 1027, 1028, 1032, 1033, 1035, and 1041-1062 are valid based on the restrictions mentioned above.

buf ___A pointer to a buffer containing the wksta information.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

The WkstaUser APIs are:

4.4.3. NetWkstaUserGetInfo

NetWkstaUserGetInfo returns the user-specific information about the configuration elements for a workstation.

Privilege Level

Privilege requirements are described with the data structures for the infolevels above. This API only works locally.

NET_API_STATUS NET_API_FUNCTION
NetWkstaUserGetInfo (
IN LPWSTR *reserved*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

reserved ___This field must be set to zero.

level ___Level of information required. 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.4.4. NetWkstaUserSetInfo

NetWkstaUserSetInfo sets the user-specific information about the configuration elements for a workstation.

Privilege Level

Privilege requirements are described with the data structures for the infolevels above. This API only works locally.

NET_API_STATUS NET_API_FUNCTION
NetWkstaUserSetInfo (
IN LPWSTR *reserved*,
IN DWORD *level*,
OUT LPBYTE * *bufptr.*
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

reserved ___This field must be set to zero.

level ___Level of information required. 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.4.5. NetWkstaUserEnum

NetWkstaUserEnum lists information about all current users on the workstation.

Privilege Level

Admin privilege is required to successfully execute **NetWkstaUserEnum** both locally and on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetWkstaUserEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information provided. Must be 0 or 1.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing use search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.5. Use APIs

The WNetAddConnection(2) and WNetCancelConnection(2) APIs should be used instead of the NetUse APIs.

Use API functions examine or control connections (uses) between workstations and servers. Connections are distinguished from sessions: a session is established the first time a workstation makes a connection to a shared resource on the server; all further connections between the workstation and the server are part of this same session until the session ends. Two types of connections can be made: devicename connections (which can only be explicit) and universal-naming convention (UNC) connections (which can be explicit or implicit).

Connections are made on a per user basis. A connection made by a user is deleted when that user logs off. For this reason the NetUse API are local only, since a connection set up by a remote user would not be accessible to any other users, even the user that was interactively logged onto that machine.

NetUseAdd creates a devicename connection or an explicit UNC connection. Implicit UNC connections are made by the function responsible for the connection.

NetUseAdd establishes an explicit connection between the local computer and a resource shared on a server by redirecting a local devicename to the sharename of a remote server resource (**\\servername\sharename**). Once a devicename connection is made, users or applications can use the remote resource by specifying the local devicename. To establish an implicit UNC connection, an application passes the sharename of a resource to any function that accepts UNC pathnames.. The function accepts the UNC name and makes a connection to the specified sharename. All further requests on this connection require the full sharename.

NetUseDel ends a connection to a shared resource. **NetUseEnum** enumerates all current connections between the local computer and resources on remote servers. **NetUseGetInfo** returns information about a connection to a shared resource.

The Use APIs are available at three information levels.

```
typedef struct _USE_INFO_0 {
        LPWSTR          ui0_local;
        LPWSTR          ui0_remote;
} USE_INFO_0, *PUSE_INFO_0, *LPUSE_INFO_0;

typedef struct _USE_INFO_1 {
        LPWSTR          ui1_local;
        LPWSTR          ui1_remote;
        LPWSTR          ui1_password;
        DWORD           ui1_status;
        DWORD           ui1_asg_type;
        DWORD           ui1_refcount;
        DWORD           ui1_usecount;
} USE_INFO_1, *PUSE_INFO_1, *LPUSE_INFO_1;
```

Infolevel 2 is not available if the API is remoted to a downlevel system - ERROR_NOT_SUPPORTED will be returned in that case.

```
typedef struct _USE_INFO_2 {
        LPWSTR          ui2_local;
        LPWSTR          ui2_remote;
        LPWSTR          ui2_password;
        DWORD           ui2_status;
        DWORD           ui2_asg_type;
        DWORD           ui2_refcount;
        DWORD           ui2_usecount;
        LPWSTR          ui2_username;
        LPWSTR          ui2_domainname;
} USE_INFO_2, *PUSE_INFO_2, *LPUSE_INFO_2;
```

The use APIs are:

4.5.1. NetUseAdd

WNetAddConnection(2) should be used in place of NetUseAdd.

NetUseAdd establishes a connection between a local or NULL devicename and a shared resource by redirecting the local or NULL (UNC) devicename to the shared resource.

NET_API_STATUS NET_API_FUNCTION
NetUseAdd (
IN LPWSTR *reserved* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

reserved __Must be NULL.

level __Level of information to set. 1 and 2 are valid.

buf __A pointer to a buffer containing the use information structure.

parm_err __Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.5.2. NetUseDel

WNetCancelConnection(2) should be used in place of NetUseDel.

NetUseDel ends a connection to a shared resource.

Privilege Level

This API cannot be executed on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetUseDel (
IN LPWSTR *reserved* **OPTIONAL,**
IN LPWSTR *usename*,
IN DWORD *ucond*
);

Parameters:

reserved __Must be NULL.

usename __A pointer to an UNICODE string containing the path of the use to delete.

ucond __Level of force to use in deleting the use.

4.5.3. NetUseEnum

WNetEnumResource should be used in place of NetUseEnum.

NetUseEnum lists all current connections between the local computer and resources on remote servers.

Privilege Level

This API cannot be executed on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetUseEnum (
IN LPWSTR *reserved* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

reserved __Must be NULL.

level __Level of information provided. Must be 0, 1 or 2.

bufptr __On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen __Prefered maximum length of returned data (in 8-bit bytes).

entriesread __On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries __On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing use search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.5.4. NetUseGetInfo

WNetGetConnection should be used in place of NetUseGetInfo.

NetUseGetInfo retrieves information about a connection to a shared resource.

NET_API_STATUS NET_API_FUNCTION
NetUseGetInfo (
IN LPWSTR *reserved* **OPTIONAL**,
IN LPWSTR *usename*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

reserved ___Must be NULL.

usename ___A pointer to an UNICODE string containing the local or remote name active use to return information on.

level ___Level of information required. 0, 1 and 2 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.6. Share APIs

Share API functions control shared resources. A shared resource is a local resource on a server (for example, a disk directory, print device, or named pipe) that can be accessed by users and applications on the network.

NetShareAdd allows a user or application to share a resource of a specific type using the specified sharename. **NetShareAdd** requires the sharename and local devicename to share the resource. A user or application must have an account on the server to access the resource.

LAN Manager defines three types of special sharenames for interprocess communication (IPC) and remote administration of the server:

- IPC$, reserved for interprocess communication.
- ADMIN$, reserved for remote administration.
- A$, B$, C$ (and other local disk names followed by a dollar sign), assigned to local disk devices.

Share APIs are available at three information levels.

```
typedef struct _SHARE_INFO_0 {
        LPWSTR          shi0_netname;
} SHARE_INFO_0, *PSHARE_INFO_0, *LPSHARE_INFO_0;

typedef struct _SHARE_INFO_1 {
        LPWSTR          shi1_netname;
        DWORD           shi1_type;
        LPWSTR          shi1_remark;
} SHARE_INFO_1, *PSHARE_INFO_1, *LPSHARE_INFO_1;


typedef struct _SHARE_INFO_2 {
        LPWSTR          shi2_netname;
        DWORD           shi2_type;
        LPWSTR          shi2_remark;
        DWORD           shi2_permissions;
        DWORD           shi2_max_uses;
        DWORD           shi2_current_uses;
        LPWSTR          shi2_path;
        LPWSTR          shi2_passwd;
} SHARE_INFO_2, *PSHARE_INFO_2, *LPSHARE_INFO_2;
```

The following infolevels are only valid for **NetShareSetInfo** and replace the older way of passing in a *Parmnum* to set a specific field.

The following are supported on downlevel systems (i.e. LANMan 2.x) as well:

```
typedef struct _SHARE_INFO_1004 {
        LPWSTR          shi1004_remark;
} SHARE_INFO_1004, *PSHARE_INFO_1004, *LPSHARE_INFO_1004;

typedef struct _SHARE_INFO_1006 {
        DWORD           shi1006_max_uses;
} SHARE_INFO_1006, *PSHARE_INFO_1006, *LPSHARE_INFO_1006;
```

For **NetShareSetInfo**, parmnum values refer to the fields in the share_info structure as follows. These values are used when indicating an error in a specific parameter via *parm_err*.

parmnum value	Field in share_info struct
SHI_NETNAME_PARMNUM	shi_netname
SHI_TYPE_PARMNUM	shi_type
SHI_REMARK_PRAMNUM	shi_remark
SHI_PERMISSIONS_PARMNUM	shi_permissions
SHI_MAX_USES_PARMNUM	shi_max_uses
SHI_CURRENT_USES_PARMNUM	shi_current_uses
SHI_PATH_PARMNUM	shi_path
SHI_PASSWD_PARMNUM	shi_passwd

The Share APIs are:

4.6.1. NetShareAdd

NetShareAdd shares a server resource.

Privilege Level

Do we still have comm operator and comm device queues?

This is now sticky, correct?

Admin privilege or comm, print, or server operator privilege is required to successfully execute **NetShareAdd** on a remote server or on a computer that has local security enabled. The print operator can add only printer queues. The comm operator can add only communication-device queues.

NET_API_STATUS NET_API_FUNCTION
NetShareAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err OPTIONAL*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information provided. Must be 2.

buf ___A pointer to a buffer containing the share information structure.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.6.2. NetShareEnum

WNetEnumResource should be used in place of NetShareEnum.

NetShareEnum retrieves information about each shared resource on a server.

Privilege Level

Admin privilege or comm, print, or server operator privilege is required to successfully execute **NetShareEnum** at level 2 on a remote server or on a computer that has local security enabled. No special privilege is required for level 0 or level 1 calls.

NET_API_STATUS NET_API_FUNCTION
NetShareEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. 0, 1 and 2 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing share search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.6.3. NetShareGetInfo

NetShareGetInfo retrieves information about a particular shared resource on a server.

Privilege Level

Admin privilege or comm, print, or server operator privilege is required to successfully execute **NetShareGetInfo** at level 2 on a remote server or on a computer that has local security enabled. No special privilege is required for level 0 or level 1 calls.

NET_API_STATUS NET_API_FUNCTION
NetShareGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *netname*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

netname ___A pointer to an UNICODE string containing the netname of the share to return information on.

level ___Level of information required. 0, 1 and 2 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.6.4. NetShareSetInfo

NetShareSetInfo sets the parameters of a shared resource.

Is this SetInfo also sticky?

Privilege Level

Admin privilege or comm, print, or server operator privilege is required to successfully execute **NetShareSetInfo** on a remote server or on a computer that has local security enabled. The print operator can set information only about printer queues. The comm operator can set information only about communication-device queues.

NET_API_STATUS NET_API_FUNCTION
NetShareSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *netname*,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err OPTIONAL*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

netname ___A pointer to an UNICODE string containing the netname of the share to set information on.

level ___Level of information to set. 1, 2, 1004 - 1006 and 1009 are valid.

buf ___A pointer to a buffer containing the share information.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.6.5. NetShareDel

NetShareDel deletes a sharename from a server's list of shared resources, disconnecting all connections to the shared resource.

Privilege Level

Admin privilege or comm, print, or server operator privilege is required to successfully execute **NetShareDel** on a remote server or on a computer that has local security enabled. The print operator can delete only printer queues. The comm operator can delete only communication-device queues.

NET_API_STATUS NET_API_FUNCTION
NetShareDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *netname*,
IN DWORD *reserved*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

netname ___A pointer to an UNICODE string containing the netname of the share to delete.

reserved ___Reserved, must be zero.

4.6.6. NetShareCheck

NetShareCheck checks whether or not a server is sharing a device.

Privilege Level

No special privilege level is required to successfully execute **NetShareCheck**.

NET_API_STATUS NET_API_FUNCTION
NetShareCheck (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *device*,
OUT LPDWORD *type*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

device ___A pointer to an UNICODE string containing the name of the device to check for shared access.

type ___On return the address pointed to by the type parameter contains the type of share the device is offered with. This field is only set if success was returned.

4.7. Session

Session API functions control network sessions established between workstations and servers. They require that the Server service be started on the specified server. A session is a link between a workstation and a server. It is established the first time a workstation makes a connection with a shared resource on the server. Until the session ends, all further connections between the workstation and the server are part of this same session. To end a session, an application on the server end of a connection calls **NetSessionDel**. This deletes all current connections between the workstation and the server. **NetSessionEnum** returns information about all sessions established for a server. **NetSessionGetInfo** returns information about a particular session.

Per-user information is managed by the NetSession APIs via the use of the *username* parameter. Since there can be multiple users per session, this parameter is necessary in order to access the user-specific information for the session.

Session APIs are available at four information levels.

```
typedef struct _SESSION_INFO_0 {
        LPWSTR          sesi0_cname;
} SESSION_INFO_0, *PSESSION_INFO_0, *LPSESSION_INFO_0;

typedef struct _SESSION_INFO_1 {
        LPWSTR          sesi1_cname;
        LPWSTR          sesi1_username;
        DWORD           sesi1_num_opens;
        DWORD           sesi1_time;
        DWORD           sesi1_idle_time;
        DWORD           sesi1__flags;
} SESSION_INFO_1, *PSESSION_INFO_1, *LPSESSION_INFO_1;

typedef struct _SESSION_INFO_2 {
        LPWSTR          sesi2_cname;
        LPWSTR          sesi2_username;
        DWORD           sesi2_num_opens;
        DWORD           sesi2_time;
        DWORD           sesi2_idle_time;
        DWORD           sesi2_flags;
        LPWSTR          sesi2_cltype_name;
} SESSION_INFO_2, *PSESSION_INFO_2, *LPSESSION_INFO_2;

typedef struct _SESSION_INFO_10 {
        LPWSTR          sesi10_cname;
        LPWSTR          sesi10_username;
        DWORD           sesi10_time;
        DWORD           sesi10_idle_time;
} SESSION_INFO_10, *PSESSION_INFO_10, *LPSESSION_INFO_10;
```

sesi_username cannot be NULL. sesi1_flags and sesi2_flags can take the following bits:

SESS_GUEST	0x00000001
SESS_NONENCRYPTION	0x00000002

The session APIs are:

4.7.1. NetSessionEnum

NetSessionEnum provides information about all current sessions.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetSessionEnum** at level 1 or level 2. No special privilege is required for level 0 or level 10 calls.

NET_API_STATUS NET_API_FUNCTION
NetSessionEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *clientname* **OPTIONAL**,
IN LPWSTR *username* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

clientname ___A pointer to an UNICODE string containing the name of the computer session for which information is to be returned. A NULL pointer or string specifies that all computer sessions on the server are to be ennumerated.

username ___A pointer to an UNICODE string containing the name of the the user for which to ennumerate the sessions. A NULL pointer or string specifies that sessions for all users are to be ennumerated.

level ___Level of information required. 0, 1, 2 and 10 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing session search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored.

4.7.2. NetSessionGetInfo

NetSessionGetInfo retrieves information about a session established between a particular server and workstation.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetSessionGetInfo** at level 1 or level 2. No special privilege is required for level 0 or level 10 calls.

NET_API_STATUS NET_API_FUNCTION
NetSessionGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *clientname*,
IN LPWSTR *username*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

clientname ___A pointer to an UNICODE string containing the name of the computer session for which information is to be returned. This field cannot be NULL.

username ___A pointer to an UNICODE string containing the name of the user whose session information is to be returned. This field cannot be NULL.

level ___Level of information required. 0, 1, 2 and 10 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.7.3. NetSessionDel

NetSessionDel ends a session between a server and a workstation.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetSessionDel**.

NET_API_STATUS NET_API_FUNCTION
NetSessionDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *clientname*,
IN LPWSTR *username*,
IN DWORD *reserved*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

clientname ___A pointer to an UNICODE string containing the computer name of the client to disconnect. If clientname is NULL, then all the sessions of the user in username will be deleted on the server specified.

username ___A pointer to an UNICODE string containing the name of the user whose session is to be terminated. A NULL indicates that all users' sessions from the clientname specified are to be terminated.

reserved ___reserved, must be zero.

4.8. Connection APIs

WNetConnection APIs should be used in place of NetConnection APIs.

The Connection API function, **NetConnectionEnum**, lists server connections. A computer accesses a shared resource on a server by means of a connection. A connection is a software link between a computer and a server, made by assigning a local or NULL devicename to the shared resource on the server. The **NetUseAdd** function establishes connections.

NetConnectionEnum lists information about all connections to a server made by a specified computer, or about all connections made to a specified sharename.

Connection APIs are available at two information levels:

```
typedef struct _CONNECTION_INFO_0 {
        DWORD                   coni0_id;
} CONNECTION_INFO_0, *PCONNECTION_INFO_0, *LPCONNECTION_INFO_0;
```

```
typedef struct _CONNECTION_INFO_1 {
        DWORD           coni1_id;
        DWORD           coni1_type;
        DWORD           coni1_num_opens;
        DWORD           coni1_num_users;
        DWORD           coni1_time;
        LPWSTR          coni1_username;
        LPWSTR          coni1_netname;
} CONNECTION_INFO_1, *PCONNECTION_INFO_1, *LPCONNECTION_INFO_1;
```

The connection APIs are:

4.8.1. NetConnectionEnum

WNetEnumResource should be used in place of NetConnectionEnum.

NetConnectionEnum lists all connections made to a shared resource on the server or all connections established from a particular computer. If there is more than one user using this connection, then it is possible to get more than one structure for the same connection, but with different username.

Privilege Level

Admin privilege or server, print, or comm operator privilege is required to successfully execute **NetConnectionEnum**.

NET_API_STATUS NET_API_FUNCTION
NetConnectionEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *qualifier*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

qualifier ___A pointer to an UNICODE string containing a sharename or computername for the connections of interest. If it is a sharename, then all the connections made to that sharename are listed. If it is a computername (i.e. it starts with two backslash characters), then *NetConnectionEnum* lists all connections made from that computer to the server specified.

level ___Level of information required. 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen __Prefered maximum length of returned data (in 8-bit bytes).

entriesread __On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries __On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle __On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing connection search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.9. File APIs

File API functions provide a way to monitor and close the file, device, and pipe resources open on a server. **NetFileClose** forces a server resource closed. This function can be used when a system error prevents normal closure. **NetFileEnum** returns information about resources open on a server. A file can be opened one or more times by one or more applications. Each file opening is uniquely identified. **NetFileEnum** returns an entry for each file opening. **NetFileGetInfo** returns information about one particular opening of a resource.

File APIs are available at information levels 2 and 3 only. Levels 0 and 1 are not supported.

```
typedef struct _FILE_INFO_2 {
        DWORD           fi2_id;
} FILE_INFO_2, *PFILE_INFO_2, *LPFILE_INFO_2;

typedef struct _FILE_INFO_3 {
        DWORD           fi3_id;
        DWORD           fi3_permissions;
        DWORD           fi3_num_locks;
        LPWSTR          fi3_pathname;
        LPWSTR          fi3_username;
} FILE_INFO_3, *PFILE_INFO_3, *LPFILE_INFO_3;
```

The file APIs are:

4.9.1. NetFileEnum

NetFileEnum supplies information about some or all open files on a server, allowing the user to supply a resume handle and get required information through repeated calls to the function.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetFileEnum**.

NET_API_STATUS NET_API_FUNCTION
NetFileEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *basepath*,
IN LPWSTR *username*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

basepath ___A pointer to an UNICODE string containing a qualifier for the returned information. If NULL then all open resources are enumerated, else only resources which have basepath as a prefix are enumerated. A *prefix* is the path component up to a back-slash.

username ___A pointer to an UNICODE string that specifies the name of the user. If not NULL, username serves as a qualifier to the ennumeration. The files returned are limited to those that have usernames matching the qualifier. If username is NULL, no username qualifier is used.

level ___Level of information required. 2 and 3 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing file search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.9.2. NetFileGetInfo

NetFileGetInfo retrieves information about a particular opening of a server resource.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetFileGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetFileGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *fileid*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

fileid ___The fileid of the open resource to return information on. The fileid value must be that returned in a previous enumeration call.

level ___Level of information required. 2 and 3 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.9.3. NetFileClose

NetFileClose forces a resource to close. This function can be used when an error prevents closure by other means.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetFileClose**.

NET_API_STATUS NET_API_FUNCTION
NetFileClose (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *fileid*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

fileid ___The fileid of the opened resource instance to be closed.

4.10. Message APIs

Message API functions send and receive messages and manipulate message aliases. A message is a buffer of text data sent to a user or application on the network. To receive a message, a user or application must register a message alias in a computer's table of message names. This can be done by using **NetMessageNameAdd**. A message name

table contains a list of registered message aliases (users and applications) permitted to receive messages.

The aliases registered in the message name table are case insensitive.
NetMessageNameDel deletes a specific message alias from the message name table.
NetMessageNameEnum lists all the aliases stored in the message name table.
NetMessageNameGetInfo retrieves information about a particular message alias in the message name table.

To send a message, an application calls **NetMessageBufferSend**.
Applications can also send broadcast messages to all users in a domain or to all computers on a network using **NetMessageBufferSend**.

Message APIs are available at two information levels.

MSG_INFO_1 only exists for compatability. The NT messenger will not forward names or allow names to be forwarded to it.

```
typedef struct _MSG_INFO_0 {
        LPWSTR          msgi0_name;
} MSG_INFO_0, *PMSG_INFO_0, *LPMSG_INFO_0;

typedef struct _MSG_INFO_1 {
        LPWSTR          msgi1_name;
        DWORD           msgi1_forward_flag;
        LPWSTR          msgi1_forward;
} MSG_INFO_1, *PMSG_INFO_1, *LPMSG_INFO_1;
```

The Message APIs are:

4.10.1. NetMessageNameAdd

NetMessageNameAdd registers a message alias in the message name table.
NetMessageNameAdd requires that the Messenger service be started.

Privilege Level

Admin privilege is required to successfully execute **NetMessageNameAdd** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetMessageNameAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *msgname*
};

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

msgname ___A pointer to an UNICODE string of limit 15 characters containing the message name to be added.

Note. The forward action flag from the LANMan 2.x **NetMessageNameAdd** is no longer a parameter as message forwarding is no longer supported. If the NetMessageNameAdd API detects that a forwarded version of msgname exists on the network then the API will fail with error NERR_Already_Exists.

4.10.2. NetMessageNameEnum

NetMessageNameEnum lists the message aliases that will receive messages on a specified computer. **NetMessageNameEnum** requires that the Messenger service be started.

Privilege Level

Admin privilege is required to successfully execute **NetMessageNameEnum** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetMessageNameEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing message name search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.10.3. NetMessageNameGetInfo

NetMessageNameGetInfo retrieves information about a particular message alias in the message name table. **NetMessageNameGetInfo** requires that the Messenger service be started.

Privilege Level

Admin privilege is required to successfully execute **NetMessageNameGetInfo** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetMessageNameGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *msgname*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

msgname ___A pointer to an UNICODE string containing the message name to return information on.

level ___Level of information required. 0 & 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.10.4. NetMessageNameDel

NetMessageNameDel deletes a message alias from the table of message aliases on a computer. **NetMessageNameDel** requires that the Messenger service be started.

Privilege Level

Admin privilege is required to successfully execute **NetMessageNameDel** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetMessageNameDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *msgname*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

msgname ___A pointer to an UNICODE string of limit 15 characters containing the message name to be deleted.

4.10.5. NetMessageBufferSend

NetMessageBufferSend sends a buffer of information to a registered message alias.

Privilege Level

No special privilege is required to execute NetMessage buffer send locally. Admin, accounts, print, or server operator privilege is required to successfully execute **NetMessageBufferSend** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetMessageBufferSend (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *msgname*,
IN LPWSTR *fromname*,
IN LPBYTE *buf*,
IN DWORD *buflen*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

msgname ___A pointer to an UNICODE string containing the message name to which the message buffer should be sent.

fromname ___A pointer to an UNICODE string containing the message name sending the information. *fromname* is a new parameter for Windows networking. This parameter is needed for sending alerts from the computer name rather than the logged on user. If a NULL is specified the message is send from the logged on user as with LANMan 2.x.

buf ___A pointer to a buffer of message text.

buflen ___The length in bytes of the message text in buf.

4.11. Remote Utility API

Remote Utility API functions enable applications to access the time-of-day information on a remote server.

NetRemoteTOD returns time-of-day information from a remote server.

4.11.1. NetRemoteTOD API

The remote time of day information is available at one information level:

```
typedef struct _TIME_OF_DAY_INFO {
        DWORD           tod_elapsedt;
        DWORD           tod_msecs;
        DWORD           tod_hours;
        DWORD           tod_mins;
        DWORD           tod_secs;
        DWORD           tod_hunds;
        LONG            tod_timezone;
        DWORD           tod_tinterval;
        DWORD           tod_day;
        DWORD           tod_month;
        DWORD           tod_year;
        DWORD           tod_weekday;
} TIME_OF_DAY_INFO, *PTIME_OF_DAY_INFO, *LPTIME_OF_DAY_INFO;
```

NetRemoteTOD returns a server's time of day.

Privilege Level

No special privilege level is required to successfully execute **NetRemoteTOD**.

NET_API_STATUS NET_API_FUNCTION
NetRemoteTOD (
IN LPWSTR *servername* **OPTIONAL**,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

4.12. Security Account APIs

4.12.1. User APIs

User API functions control a user's account in the security database. Each user or application that accesses resources must have an account in the security database. The NT Security system (SAM) uses this user account to verify that the user or application has permission to use a resource. When a user or an application requests access to a resource, the security system checks for an appropriate user account or group account to permit the access.

NetUserEnum can be used to list all user accounts in a domain. An application can change a user's privilege level by calling **NetUserSetInfo**. It can also change the user's resource access privileges by modifying that user's groups (for more information, see the Group category API functions). Individually assigned user privileges take precedence over group privileges. An application can verify the groups to which a user belongs by calling **NetUserGetGroups**, which returns a list of global groupnames. The **NetUserGetLocalGroups** function does the same for local groups. When a user account is no longer needed, use **NetUserDel** to delete the account from the server. Once the

account is removed, the user can no longer access the server, except by using the **guest** account. Because the user's password is confidential, it is not returned by **NetUserEnum** or **NetUserGetInfo**. The password is initially assigned when **NetUserAdd** is called. **NetUserSetInfo** sets the password and other elements of a user account.

User account information is available at five levels:

```
typedef struct _USER_INFO_0 {
        LPWSTR          usri0_name;
} USER_INFO_0, *PUSER_INFO_0, *LPUSER_INFO_0;

typedef struct _USER_INFO_1 {
        LPWSTR          usri1_name;
        LPWSTR          usri1_password;
        DWORD           usri1_password_age;
        DWORD           usri1_priv;
        LPWSTR          usri1_home_dir;
        LPWSTR          usri1_comment;
        DWORD           usri1_flags;
        LPWSTR          usri1_script_path;
} USER_INFO_1, *PUSER_INFO_1, *LPUSER_INFO_1;

typedef struct _USER_INFO_2 {
        LPWSTR          usri2_name;
        LPWSTR          usri2_password;
        DWORD           usri2_password_age;
        DWORD           usri2_priv;
        LPWSTR          usri2_home_dir;
        LPWSTR          usri2_comment;
        DWORD           usri2_flags;
        LPWSTR          usri2_script_path;
        DWORD           usri2_auth_flags;
        LPWSTR          usri2_full_name;
        LPWSTR          usri2_usr_comment;
        LPWSTR          usri2_parms;
        LPWSTR          usri2_workstations;
        DWORD           usri2_last_logon;
        DWORD           usri2_last_logoff;
        DWORD           usri2_acct_expires;
        DWORD           usri2_max_storage;
        DWORD           usri2_units_per_week;
        LPBYTE          usri2_logon_hours;
        DWORD           usri2_bad_pw_count;
        DWORD           usri2_num_logons;
        LPWSTR          usri2_logon_server;
        DWORD           usri2_country_code;
        DWORD           usri2_code_page;
} USER_INFO_2, *PUSER_INFO_2, *LPUSER_INFO_2;

typedef struct _USER_INFO_3 {
        LPWSTR          usri3_name;
        LPWSTR          usri3_password;
        DWORD           usri3_password_age;
        DWORD           usri3_priv;
```

```
        LPWSTR          usri3_home_dir;
        LPWSTR          usri3_comment;
        DWORD           usri3_flags;
        LPWSTR          usri3_script_path;
        DWORD           usri3_auth_flags;
        LPWSTR          usri3_full_name;
        LPWSTR          usri3_usr_comment;
        LPWSTR          usri3_parms;
        LPWSTR          usri3_workstations;
        DWORD           usri3_last_logon;
        DWORD           usri3_last_logoff;
        DWORD           usri3_acct_expires;
        DWORD           usri3_max_storage;
        DWORD           usri3_units_per_week;
        PBYTE           usri3_logon_hours;
        DWORD           usri3_bad_pw_count;
        DWORD           usri3_num_logons;
        LPWSTR          usri3_logon_server;
        DWORD           usri3_country_code;
        DWORD           usri3_code_page;
        DWORD           usri3_user_id;
        DWORD           usri3_primary_group_id;
        LPWSTR          usri3_profile;
        LPWSTR          usri3_home_dir_drive;
        DWORD           usri3_password_expired;
}USER_INFO_3, *PUSER_INFO_3, *LPUSER_INFO_3;

typedef struct _USER_INFO_10 {
        LPWSTR          usri10_name;
        LPWSTR          usri10_comment;
        LPWSTR          usri10_usr_comment;
        LPWSTR          usri10_full_name;
} USER_INFO_10, *PUSER_INFO_10, *LPUSER_INFO_10;
```

```
typedef struct _USER_INFO_11 {
        LPWSTR          usri11_name;
        LPWSTR          usri11_comment;
        LPWSTR          usri11_usr_comment;
        LPWSTR          usri11_full_name;
        DWORD           usri11_priv;
        DWORD           usri11_auth_flags;
        DWORD           usri11_password_age;
        LPWSTR          usri11_home_dir;
        LPWSTR          usri11_parms;
        DWORD           usri11_last_logon;
        DWORD           usri11_last_logoff;
        DWORD           usri11_bad_pw_count;
        DWORD           usri11_num_logons;
        LPWSTR          usri11_logon_server;
        DWORD           usri11_country_code;
        LPWSTR          usri11_workstations;
        DWORD           usri11_max_storage;
        DWORD           usri11_units_per_week;
        LPBYTE          usri11_logon_hours;
        DWORD           usri11_code_page;
} USER_INFO_11, *PUSER_INFO_11, *LPUSER_INFO_11;

typedef struct _USER_INFO_20 {
        LPWSTR          usri20_name;
        LPWSTR          usri20_full_name;
        LPWSTR          usri20_comment;
        DWORD           usri20_flags;
        DWORD           usri20_user_id;
}USER_INFO_20, *PUSER_INFO_20, *LPUSER_INFO_20;
```

The following infolevels are only valid for **NetUserSetInfo** and replace the older way of passing in a *Parmnum* to set a specific field.

The following are supported on downlevel systems (i.e. LANMan 2.x) as well:

```
typedef struct _USER_INFO_1003 {
        LPWSTR          usri1003_password;
} USER_INFO_1003, *PUSER_INFO_1003, *LPUSER_INFO_1003;

typedef struct _USER_INFO_1005 {
        DWORD           usri1005_priv;
} USER_INFO_1005, *PUSER_INFO_1005, *LPUSER_INFO_1005;

typedef struct _USER_INFO_1006 {
        LPWSTR          usri1006_home_dir;
} USER_INFO_1006, *PUSER_INFO_1006, *LPUSER_INFO_1006;

typedef struct _USER_INFO_1007 {
        LPWSTR          usri1007_comment;
} USER_INFO_1007, *PUSER_INFO_1007, *LPUSER_INFO_1007;
```

```
typedef struct _USER_INFO_1008 {
        DWORD           usri1008_flags;
} USER_INFO_1008, *PUSER_INFO_1008, *LPUSER_INFO_1008;

typedef struct _USER_INFO_1009 {
        LPWSTR          usri1009_script_path;
} USER_INFO_1009, *PUSER_INFO_1009, *LPUSER_INFO_1009;

typedef struct _USER_INFO_1010 {
        DWORD           usri1010_auth_flags;
} USER_INFO_1010, *PUSER_INFO_1010, *LPUSER_INFO_1010;

typedef struct _USER_INFO_1011 {
        LPWSTR          usri1011_full_name;
} USER_INFO_1011, *PUSER_INFO_1011, *LPUSER_INFO_1011;

typedef struct _USER_INFO_1012 {
        LPWSTR          usri1012_usr_comment;
} USER_INFO_1012, *PUSER_INFO_1012, *LPUSER_INFO_1012;

typedef struct _USER_INFO_1013 {
        LPWSTR          usri1013_parms;
} USER_INFO_1013, *PUSER_INFO_1013, *LPUSER_INFO_1013;

typedef struct _USER_INFO_1014 {
        LPWSTR          usri1014_workstations;
} USER_INFO_1014, *PUSER_INFO_1014, *LPUSER_INFO_1014;

typedef struct _USER_INFO_1017 {
        DWORD           usri1017_acct_expires;
} USER_INFO_1017, *PUSER_INFO_1017, *LPUSER_INFO_1017;

typedef struct _USER_INFO_1018 {
        DWORD           usri1018_max_storage;
} USER_INFO_1018, *PUSER_INFO_1018, *LPUSER_INFO_1018;

typedef struct _USER_INFO_1020 {
        DWORD           usri1020_units_per_week;
        LPBYTE          usri1020_logon_hours;
} USER_INFO_1020, *PUSER_INFO_1020, *LPUSER_INFO_1020;

typedef struct _USER_INFO_1023 {
        LPWSTR          usri1023_logon_server;
} USER_INFO_1023, *PUSER_INFO_1023, *LPUSER_INFO_1023;

typedef struct _USER_INFO_1024 {
        DWORD           usri1024_country_code;
} USER_INFO_1024, *PUSER_INFO_1024, *LPUSER_INFO_1024;

typedef struct _USER_INFO_1025 {
        DWORD           usri1025_code_page;
} USER_INFO_1025, *PUSER_INFO_1025, *LPUSER_INFO_1025;

typedef struct _USER_INFO_1051 {
```

```
        DWORD           usri1051_primary_group_id;
} USER_INFO_1051, *PUSER_INFO_1051, *LPUSER_INFO_1051;

typedef struct _USER_INFO_1052 {
        LPWSTR          usri1052_profile;
} USER_INFO_1052, *PUSER_INFO_1052, *LPUSER_INFO_1052;

typedef struct _USER_INFO_1053 {
        LPWSTR          usri1053_home_dir_drive;
} USER_INFO_1053, *PUSER_INFO_1053, *LPUSER_INFO_1053;
```

For **NetUserSetInfo**, parmnum values refer to the fields in the user_info structure as follows. These values are used when indicating an error in a specific parameter via *parm_err*.

parmnum value	Field in user_info struct
USER_NAME_PARMNUM	usri_name
USER_PASSWORD_PARMNUM	usri_password
USER_PASSWORD_AGE_PARMNUM	usri_password_age
USER_PRIV_PARMNUM	usri_priv
USER_HOME_DIR_PARMNUM	usri_home_dir
USER_COMMENT_PARMNUM	usri_comment
USER_FLAGS_PARMNUM	usri_flags
USER_SCRIPT_PATH_PARMNUM	usri_script_path
USER_AUTH_FLAGS_PARMNUM	usri_auth_flags
USER_FULL_NAME_PARMNUM	usri_full_name
USER_USR_COMMENT_PARMNUM	usri_usr_comment
USER_PARMS_PARMNUM	usri_parms
USER_WORKSTATIONS_PARMNUM	usri_workstations
USER_LAST_LOGON_PARMNUM	usri_last_logon
USER_LAST_LOGOFF_PARMNUM	usri_last_logoff
USER_ACCT_EXPIRES_PARMNUM	usri_acct_expires
USER_MAX_STORAGE_PARMNUM	usri_max_storage
USER_UNITS_PER_WEEK_PARMNUM	usri_units_per_week
USER_LOGON_HOURS_PARMNUM	usri_logon_hours
USER_PAD_PW_COUNT_PARMNUM	usri_bad_pw_count
USER_NUM_LOGONS_PARMNUM	usri_num_logons
USER_LOGON_SERVER_PARMNUM	usri_logon_server
USER_COUNTRY_CODE_PARMNUM	usri_country_code
USER_CODE_PAGE_PARMNUM	usri_code_page
USER_PRIMARY_GROUP_PARMNUM	usri_primary_group_id
USER_PROFILE	usri_profile
USER_HOME_DIR_DRIVE_PARMNUM	usri_home_dir_drive

4.12.1.1. NetUserAdd

NetUserAdd adds a user account.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserAdd**.

NET_API_STATUS NET_API_FUNCTION
NetUserAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information provided. Must be 1, 2 or 3.

buf ___A pointer to a buffer containing the user information structure.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.1.2. NetUserEnum

NetUserEnum provides resumable enumeration of information about each user account in a domain.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserEnum** at levels 1 and 2. No special privilege is required at level 0 or 10.

NET_API_STATUS NET_API_FUNCTION
NetUserEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN DWORD *filter*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. 0, 1, 2, 3, 10, 11 and 20 are valid.

*filter*__Specifies a filter of account types ot enumerate. 0 implies all account types. Allowable values are:

FILTER_TEMP_DUPLICATE_ACCOUNTS
FILTER_NORMAL_ACCOUNT
FILTER_INTERDOMAIN_TRUST_ACCOUNT
FILTER_WORKSTATION_TRUST_ACCOUNT
FILTER_SERVER_TRUST_ACCOUNT

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing user search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.12.1.3. NetUserGetInfo

NetUserGetInfo retrieves information about a particular user account on a server.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserGetInfo** at level 1 and 2. No special privilege is required at levels 0 or 10. A user may call NetUserGetInfo on his/her own account at level 11.

NET_API_STATUS NET_API_FUNCTION
NetUserGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *username*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

username ___A pointer to an UNICODE string containing the name of the user account on which to return information.

level ___Level of information required. 0, 1, 2, 3,10,11 and 20 are valid.

bufptr ___A pointer to the return information structure is returned in the address pointed to by bufptr.

4.12.1.4. NetUserSetInfo

NetUserSetInfo sets the parameters of a user account.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserSetInfo.** A user may call **NetUserSetInfo** to set certain information on his/her own account.

NET_API_STATUS NET_API_FUNCTION
NetUserSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *username*,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

username ___A pointer to an UNICODE string containing the name of the user account to set information on.

level ___Level of information to set. 0, 1, 2, 3, or 20.

buf ___A pointer to a buffer containing the user information.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.1.5. NetUserDel

NetUserDel deletes a user account from the accounts database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserDel**.

NET_API_STATUS NET_API_FUNCTION
NetUserDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *username*
);
Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

username ___A pointer to an UNICODE string containing the name of the user account to delete

4.12.1.6. NetUserGetGroups

NetUserGetGroups retrieves a list of global groups to which a specified user belongs.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserGetGroups.**

NET_API_STATUS NET_API_FUNCTION
NetUserGetGroups (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *username*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*
);
Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

username ___A pointer to an UNICODE string containing the name of the user to return global group membership for.

level ___Level of information required. Only 0 or 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr. The returned information is an array of GROUP_USERS_INFO_x structures..

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries __On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

4.12.1.7. NetUserGetLocalGroups

NetUserGetLocalGroups retrieves a list of local groups to which a specified user belongs.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserGetLocalGroups.**

NET_API_STATUS NET_API_FUNCTION
NetUserGetLocalGroups (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *username*,
IN DWORD *level*,
IN DWORD flags,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*
);

Parameters:

servername __A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

username __A pointer to an UNICODE string containing the name of the user to return global group membership for.

level __Level of information required. Only 0 is valid.

flags __Bitmask of flags. Currently, only LG_INCLUDE_INDIRECT is defined. If this bit is set, the function will also return localgroups the user is a member of indirectly (ie. by the virtue of being in a globalgroup that itself is a member of one or more localgroups).

bufptr __On return a pointer to the return information structure is returned in the address pointed to by bufptr. The returned information is an array of LOCALGROUP_USERS_INFO_0 structures..

prefmaxlen __Prefered maximum length of returned data (in 8-bit bytes).

entriesread __On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries __On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

4.12.1.8. NetUserSetGroups

NetUserSetGroups sets global group memberships for a specified user account.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserSetGroups.**

NET_API_STATUS NET_API_FUNCTION
NetUserSetGroups (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *username*,
IN DWORD *level*,
IN LPBYTE *bufptr*,
IN DWORD *num_entries*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

username ___A pointer to an UNICODE string containing the name of the user to set global group memberships for.

level ___Level of information required. Only 0 or 1 are valid.

bufptr ___A pointer to an array of global groups information structures.

*num_entries*___Number of global group information structures contained in the array pointed to by bufptr.

4.12.2. User Modal APIS

The user modal API functions control the system wide parameters which affect the NT Security system behaviour.

User modal information is available at three levels

```
typedef struct _USER_MODALS_INFO_0 {
        DWORD           usrmod0_min_passwd_len;
        DWORD           usrmod0_max_passwd_age;
        DWORD           usrmod0_min_passwd_age;
        DWORD           usrmod0_force_logoff;
        DWORD           usrmod0_password_hist_len;
} USER_MODALS_INFO_0, *PUSER_MODALS_INFO_0,
LPUSER_MODALS_INFO_0;

typedef struct _USER_MODALS_INFO_1 {
        DWORD           usrmod1_role;
        LPWSTR          usrmod1_primary;
} USER_MODALS_INFO_1, *PUSER_MODALS_INFO_1,
*LPUSER_MODALS_INFO_1;
```

```
typedef struct _USER_MODALS_INFO_2 {
        LPWSTR          usrmod2_domain_name;
        PSID            usrmod2_domain_id;
}USER_MODALS_INFO_2, *PUSER_MODALS_INFO_2,
*LPUSER_MODALS_INFO_2;
```

The following infolevels are only valid for **NetUserModalsSet** and replace the older way of passing in a *Parmnum* to set a specific field.

The following are supported on downlevel systems (i.e. LANMan 2.x) as well. NOTE that the mapping from the old *parmnum* to the infolevel does not apply to the *role* and *primary* fields.

```
typedef struct _USER_MODALS_INFO_1001 {
        DWORD           usrmod1001_min_passwd_len;
} USER_MODALS_INFO_1001, *PUSER_MODALS_INFO_1001,
*LPUSER_MODALS_INFO_1001;

typedef struct _USER_MODALS_INFO_1002 {
        DWORD           usrmod1002_max_passwd_age;
} USER_MODALS_INFO_1002, *PUSER_MODALS_INFO_1002,
*LPUSER_MODALS_INFO_1002;

typedef struct _USER_MODALS_INFO_1003 {
        DWORD           usrmod1003_min_passwd_age;
} USER_MODALS_INFO_1003, *PUSER_MODALS_INFO_1003,
*LPUSER_MODALS_INFO_1003;

typedef struct _USER_MODALS_INFO_1004 {
        DWORD           usrmod1004_force_logoff;
} USER_MODALS_INFO_1004, *PUSER_MODALS_INFO_1004,
*LPUSER_MODALS_INFO_1004;

typedef struct _USER_MODALS_INFO_1005 {
        DWORD           usrmod1005_password_hist_len;
} USER_MODALS_INFO_1005, *PUSER_MODALS_INFO_1005,
*LPUSER_MODALS_INFO_1005;

typedef struct _USER_MODALS_INFO_1006 {
        DWORD           usrmod1006_role;
} USER_MODALS_INFO_1006, *PUSER_MODALS_INFO_1006,
*LPUSER_MODALS_INFO_1006;

typedef struct _USER_MODALS_INFO_1007 {
        LPWSTR          usrmod1007_primary;
} USER_MODALS_INFO_1007, *PUSER_MODALS_INFO_1007,
*LPUSER_MODALS_INFO_1007;
```

For **NetUserModalsSet**, parmnum values refer to the fields in the modals_info structure as follows. These values are used when indicating an error in a specific parameter via *parm_err*.

parmnum value	Field in modals_info struct

MODALS_MIN_PASSWD_LEN_PARMNUM	usrmod_min_passwd_len
MODALS_MAX_PASSWD_AGE_PARMNUM	usrmod_max_passwd_age
MODALS_MIN_PASSWD_AGE_PARMNUM	usrmod_min_passwd_age
MODALS_FORCE_LOGOFF_PARMNUM	usrmod_force_logoff
MODALS_PASSWD_HIST_LIST_PARMNUM	usrmod_passwd_hist_list
MODALS_ROLE_PARMNUM	usrmod_role
MODALS_PRIMARY_PARMNUM	usrmod_primary
MODALS_DOMAIN_NAME_PARMNUM	usrmod_domain_name
MODALS_DOMAIN_ID_PARMNUM	usrmod_domain_id

The User Modal APIs are:

4.12.2.1. NetUserModalsGet

NetUserModalsGet retrieves global information for all users and global groups in the user account database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserModalsGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetUserModalsGet (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

*level*___Level of information required. 0 1, and 2 are valid.

bufptr ___A pointer to the return information structure is returned in the address pointed to by bufptr.

4.12.2.2. NetUserModalsSet

NetUserModalsSet sets global information for all users and global groups in the user account database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetUserModalsSet.**

NET_API_STATUS NET_API_FUNCTION
NetUserModalsSet (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

username ___A pointer to an UNICODE string containing the name of the user account to set information on.

level ___Level of information to set. 0, 1 and 2 are valid.

buf ___A pointer to a buffer containing the user modals information.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.3. Global Group APIs

The global group API functions control global groups of users in a way that can be used across domains. A global group is a set of users who share common permissions in the security database. Group API functions create or delete global groups, and review or adjust the membership of the global groups. The global group has a groupname that specifies the usernames of global group members.

To create a global group, an application calls **NetGroupAdd**, supplying a groupname. Initially, the global group has no members. To assign members to the global group, call **NetGroupSetUsers**. To add a user to an existing global group, call **NetGroupAddUser**. To set general information about the global group, call **NetGroupSetInfo**.

NetGroupDelUser deletes a specified username from a global group, and **NetGroupDel** disbands a global group. **NetGroupDel** works whether or not the global group has any members.

Three Group category API functions retrieve information about the global groups on a server: **NetGroupEnum** produces a list of all global groups; **NetGroupGetUsers** lists all members of a specified global group; and **NetGroupGetInfo** returns general information about the global group.
Each user account automatically belongs to one of the special global groups **Domain Users** or **None**, according to the user's privilege level. Membership of these global groups is indirectly controlled by the **NetUserAdd**, **NetUserDel**, and **NetUserSetInfo** functions. For more information, see the User category API functions.

Group API functions control groups of users.

Group account information is available at two levels:

```
typedef struct _GROUP_INFO_0 {
        LPWSTR          grpi0_name;
} GROUP_INFO_0, *PGROUP_INFO_0, *LPGROUP_INFO_0;

typedef struct _GROUP_INFO_1 {
        LPWSTR          grpi1_name;
        LPWSTR          grpi1_comment;
} GROUP_INFO_1, *PGROUP_INFO_1, *LPGROUP_INFO_1;

typedef struct _GROUP_INFO_2 {
        LPWSTR          grpi2_name;
        LPWSTR          grpi2_comment;
        DWORD           grpi2_group_id;
        DWORD           grpi2_attributes;
}GROUP_INFO_2, *PGROUP_INFO_2;
```

The Groups to which a user belongs may be obtained at one information level.

```
typedef struct _GROUP_USERS_INFO_0 {
        LPWSTR          grui0_name;
} GROUP_USERS_INFO_0, *PGROUP_USERS_INFO_0,
*LPGROUP_USERS_INFO_0;

typedef struct _GROUP_USERS_INFO_1 {
        LPWSTR          grui1_name;
        DWORD           grui1_attributes;
} GROUP_USERS_INFO_1, *PGROUP_USERS_INFO_1,
*LPGROUP_USERS_INFO_1;
```

The following infolevels are only valid for **NetGroupSetInfo** and replace the older way of passing in a *Parmnum* to set a specific field.

The following are supported on downlevel systems (i.e. LANMan 2.x) as well:

```
typedef struct _GROUPINFO_1002 {
        LPWSTR          grpi1002_comment;
} GROUP_USERS_INFO_1002, *PGROUP_USERS_INFO_1002,
*LPGROUP_USERS_INFO_1002;

typedef struct _GROUP_INFO_1005 {
        DWORD           grpi1005_attributes;
} GROUP_INFO_1005, *PGROUP_INFO_1005, *LPGROUP_INFO_1005;
```

For **NetGroupSetInfo**, parmnum values refer to the fields in the group_info structure as follows. These values are used when indicating an error in a specific parameter via *parm_err*.

parmnum value	Field in group_info struct
GROUP_NAME_PARMNUM	grpi_name
GROUP_COMMENT_PARMNUM	grpi_comment
GROUP_ATTRIBUTES_PARMNUM	grpi_attributes

The Group APIs are:

4.12.3.1. NetGroupAdd

NetGroupAdd creates a global group in the security database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupAdd**.

NET_API_STATUS NET_API_FUNCTION
NetGroupAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information provided. Must be 0 1, or 2.

buf ___A pointer to a buffer containing the global group information structure.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.3.2. NetGroupAddUser

NetGroupAddUser gives an existing user account membership in an existing global group.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupAddUser**.

NET_API_STATUS NET_API_FUNCTION
NetGroupAddUser (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *GroupName*,
IN LPWSTR *username*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

GroupName ___A pointer to an UNICODE string containing the name of the global group to which the user is to be given membership.

Username ___A pointer to an UNICODE string containing the name of the user to be given global group membership.

4.12.3.3. NetGroupEnum

NetGroupEnum retrieves information about each global group account.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupEnum**.

NET_API_STATUS NET_API_FUNCTION
NetGroupEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing global group search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.12.3.4. NetGroupGetInfo

NetGroupGetInfo retrieves information about a particular global group account on a server.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetGroupGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the global group account to return information on.

level ___Level of information required. 0, 1 and 2 are valid.

bufptr ___A pointer to the return information structure is returned in the address pointed to by bufptr.

4.12.3.5. NetGroupSetInfo

NetGroupSetInfo sets the parameters of a global group account.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupSetInfo.**

NET_API_STATUS NET_API_FUNCTION
NetGroupSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the global group account to set information on.

level ___Level of information to set. Only 0, 1 and 2 are valid.

buf ___A pointer to a buffer containing the global group information.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.3.6. NetGroupDel

NetGroupDel deletes a global group account from the accounts database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupDel**.

NET_API_STATUS NET_API_FUNCTION
NetGroupDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the global group account to delete

4.12.3.7. NetGroupDelUser

NetGroupDelUser removes a user from a particular global group in the user account database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupDelUser**.

NET_API_STATUS NET_API_FUNCTION
NetGroupDelUser (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *GroupName*,
IN LPWSTR *Username*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

GroupName ___A pointer to an UNICODE string containing the name of the global group from which the user membership is to be removed.

Username ___A pointer to an UNICODE string containing the name of the user to remove from the global group.

4.12.3.8. NetGroupGetUsers

NetGroupGetUsers retrieves a list of the members of a particular global group in the user account database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupGetUsers** except when the request is made by a user who has membership in the specified global group in which case no special privilege is required.

NET_API_STATUS NET_API_FUNCTION
NetGroupGetUsers (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the global group whose members are to be listed.

level ___Level of information required. Only 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing usergroup search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.12.3.9. NetGroupSetUsers

NetGroupSetUsers sets the global group membership for the specified global group. Each user specified is made a member of the global group. Users that are not specified but are currently members of the global group will have their membership revoked.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetGroupSetUsers** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetGroupSetUsers (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN DWORD *level*,
IN LPBYTE *buf*,
IN DWORD *totalentries*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the global group to which the specified users belong.

level ___Level of information supplied. Only 0 is valid.

buf ___Points to the buffer in which the data to be set is stored. This data consists of a sequence of _GROUP_USERS_INFO_0 data structures.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

4.12.4. Local Group APIs

A local group is a set of users who share common permissions in the security database. A local group can have members which are either users or global groups (global groups can only contain users). The local group API functions control members of local groups in a way that can only be used by the systems within a "cluster". A "cluster" is the individual workstation if the system is a Windows NT system, but it contains all the NT Advanced Servers of a domain if the system is a NT Advanced Server. Thus, a local group defined on a workstation can only be used on that workstation, but a local group defined on an Advanced Server can be used by any other Advanced Server within the same domain. The local group API functions create or delete local groups, and review or adjust the memberships of local groups.

A member can be added to a local group by specifying the security identifier (SID) of the member. The Win32 **LookupAccountName** API can be used to translate a member account name to a SID.

To create a local group, an application calls **NetLocalGroupAdd**, supplying a local group name. Initially, the local group has no members. To assign members to the local group, call **NetLocalGroupSetMembers**. To add a member to an existing local group, call **NetLocalGroupAddMember**. To set general information about the local group, call **NetLocalGroupSetInfo**.

NetLocalGroupDelMember deletes a specified member from a local group, and **NetLocalGroupDel** disbands a local group, deleting all existing members of the local group first.

Three local group category API functions retrieve information about the local groups on a server: **NetLocalGroupEnum** produces a list of all local groups; **NetLocalGroupGetMembers** lists all members of a specified local group; and **NetLocalGroupGetInfo** returns general information about the local group.

Group account information is available at three levels:

```
typedef struct _LOCALGROUP_INFO_0 {
        LPWSTR          lgrpi0_name;
} LOCALGROUP_INFO_0, *PLOCALGROUP_INFO_0,
LPLOCALGROUP_INFO_0;

typedef struct _LOCALGROUP_INFO_1 {
        LPWSTR          lgrpi1_name;
        LPWSTR          lgrpi1_comment;
} LOCALGROUP_INFO_1, *PLOCALGROUP_INFO_1,
LPLOCALGROUP_INFO_1;

typedef struct _LOCALGROUP_INFO_1002 {
        LPWSTR          lgrpi1002_comment;
} LOCALGROUP_INFO_1002, *PLOCALGROUP_INFO_1002,
LPLOCALGROUP_INFO_1002;
```

The local group to which a member belongs may be obtained at two information level.

```
typedef struct _LOCALGROUP_MEMBERS_INFO_0 {
        PSID            lgrmi0_sid;
} LOCALGROUP_MEMBERS_INFO_0, *PLOCALGROUP_MEMBERS_INFO_0,
*LPLOCALGROUP_MEMBERS_INFO_0;

typedef struct _LOCALGROUP_MEMBERS_INFO_0 {
        PSID            lgrmi0_sid;
} LOCALGROUP_MEMBERS_INFO_0, *PLOCALGROUP_MEMBERS_INFO_0,
*LPLOCALGROUP_MEMBERS_INFO_0;

typedef struct _LOCALGROUP_MEMBERS_INFO_1 {
        PSID            lgrmi1_sid;
        SID_NAME_USE    lgrmi1_sidusage;
        LPWSTR          lgrmi1_name;
} LOCALGROUP_MEMBERS_INFO_1, *PLOCALGROUP_MEMBERS_INFO_1,
*LPLOCALGROUP_MEMBERS_INFO_1;

typedef struct _LOCALGROUP_USERS_INFO_0 {
        LPWSTR                  lgrui0_name;
```

} LOCALGROUP_USERS_INFO_0, *PLOCALGROUP_USERS_INFO_0, *LPLOCALGROUP_USERS_INFO_0;

For **NetLocalGroupSetInfo**, parmnum values refer to the fields in the group_info structure as follows. These values are used when indicating an error in a specific parameter via *parm_err*.

parmnum value	Field in group_info struct
LOCALGROUP_NAME_PARMNUM	lgrpi_name
LOCALGROUP_COMMENT_PARMNUM	lgrpi_comment

The Local Group APIs are:

4.12.4.1. NetLocalGroupAdd

NetLocalGroupAdd creates a local group in the security database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupAdd**.

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information provided. Must be 0 or 1.

buf ___A pointer to a buffer containing the local group information structure.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.4.2. NetLocalGroupAddMember

NetLocalGroupAddMember gives an existing user account or global group membership in an existing local group.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupAddMember**.

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupAddMember (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN PSID *membersid*
);
Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the local group to which the user or global group is to be given membership.

membersid ___A pointer to the SID of a user or global group from the local, primary, or trust domains to be given local group membership.

4.12.4.3. NetLocalGroupEnum

NetLocalGroupEnum retrieves information about each local group account.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupEnum**.

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);
Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing local group search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.12.4.4. NetLocalGroupGetInfo

NetLocalGroupGetInfo retrieves information about a particular local group account on a server.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the local group account to return information on.

level ___Level of information required. 0 and 1 are valid.

bufptr ___A pointer to the return information structure is returned in the address pointed to by bufptr.

4.12.4.5. NetLocalGroupSetInfo

NetLocalGroupSetInfo sets the parameters of a local group.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupSetInfo.**

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the local group account to set information on.

level ___Level of information to set. 0, 1, and 1002 are valid.

buf ___A pointer to a buffer containing the local group information.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.4.6. NetLocalGroupDel

NetLocalGroupDel deletes a local group account and all its members from the accounts database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupDel**.

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the local group account to delete

4.12.4.7. NetLocalGroupDelMember

NetLocalGroupDelMember removes a member from a particular local group in the security database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupDelMember**.

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupDelMember (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN PSID *membersid*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the local group from which the user membership is to be removed.

membersid ___A pointer to a SID of a user or global group to remove from the local group.

4.12.4.8. NetLocalGroupGetMembers

NetLocalGroupGetMembers retrieves a list of the members of a particular local group in the security database.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupGetMembers** except when the request is made by a user who has membership in the specified local group in which case no special privilege is required.

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupGetMembers (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the local group whose members are to be listed.

level ___Level of information required. Levels 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing usergroup search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.12.4.9. NetLocalGroupSetMembers

NetLocalGroupSetMembers sets the local group membership for the specified local group. Each user or global group specified is made a member of the local group. Users or global groups that are not specified but are currently members of the local group will have their membership revoked.

Privilege Level

Admin privilege or account operator privilege is required to successfully execute **NetLocalGroupSetMembers** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetLocalGroupSetMembers (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *groupname*,
IN DWORD *level*,
IN LPBYTE *buf*,
IN DWORD *totalentries*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

groupname ___A pointer to an UNICODE string containing the name of the local group to which the specified users or global groups belong.

level ___Level of information supplied. Only 0 is valid.

buf ___Points to the buffer in which the data to be set is stored. This data consists of a sequence of LOCALGROUP_MEMBERS_INFO_0 data structures.

totalentries __On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

4.12.5. Access APIs

A full set of Access setting APIs exist in Win32. These should be used in place of the NetAccess APIs. The NetAccess APIs only work when remoted to a downlevel system.

Access Permissions API functions examine or modify user or group access permissions for specified resources. An access control list (ACL) contains the name of a resource, an audit attribute field, and a list of access control entries. An access control entry (ACE) is a username or groupname and its corresponding access permissions.

The audit attribute field defines what type of events will be audited for that resource. It is possible to audit various types of events, depending on whether the resource is a file or a directory. It is possible to audit events such as opening, writing, and deleting a file; creating or deleting a directory; and changing the ACL of the resource.

NetAccessAdd creates an ACL for a resource and sets username or groupname access permissions. **NetAccessDel** deletes the ACL for a resource. **NetAccessGetInfo** returns the ACL for a particular resource. **NetAccessEnum** returns information about all ACLs.

Only users or applications with admin privilege or special permission for the resource can define or examine access permissions on a remote server or on a computer that has local security. Users have special permissions for a resource when they are granted ACCESS_PERM permission for that resource; this is also known as P permission.

NetAccessCheck verifies whether a user has permission to perform a specified operation on a particular resource. If access permission is needed, you can use **NetAccessSetInfo** to change the ACL. **NetAccessGetUserPerms** returns a specified user's or group's permission for a specified resource.

Access Permission information is available at two levels:

```
typedef struct _ACCESS_INFO_0 {
        LPWSTR          acc0_resource_name;
} ACCESS_INFO_0, *PACCESS_INFO_0, *LPACCESS_INFO_0;

typedef struct _ACCESS_INFO_1 {
        LPWSTR          acc1_resource_name;
        DWORD           acc1_attr;
        DWORD           acc1_count;
} ACCESS_INFO_1, *PACCESS_INFO_1, *LPACCESS_INFO_1;
```

Access list information is available at one levels

```
typedef struct _ACCESS_LIST {
        LPWSTR          acl_ugname;
        DWORD           acl_access;
} ACCESS_LIST, *PACCESS_LIST, *LPACCESS_LIST;
```

The following infolevels are only valid for **NetAccessSetInfo** and replace the older way of passing in a *Parmnum* to set a specific field.

The following are supported on downlevel systems (i.e. LANMan 2.x) as well:

```
typedef struct _ACCESS_INFO_1002 {
        DWORD                acc1002_attr;
} ACCESS_INFO_1002, *PACCESS_INFO_1002, *LPACCESS_INFO_1002;
```

For **NetAccessSetInfo**, parmnum values refer to the fields in the access_info structure as follows. These values are used when indicating an error in a specific parameter via *parm_err*.

parmnum value	Field in access_info struct
ACCESS_RESOURCE_NAME_PARMNUM	acc_resource_name
ACCESS_ATTR_PARMNUM	acc_attr
ACCESS_COUNT_PARMNUM	acc_count
ACCESS_ACCESS_LIST_PARMNUM	array following acc1_count

The following are defined for the bits in the acc1_attr field of _ACCESS_INFO_1:

ACCESS_AUDIT	0x001
ACCESS_SUCCESS_OPEN	0x010
ACCESS_SUCCESS_WRITE	0x020
ACCESS_SUCCESS_DELETE	0x040
ACCESS_SUCCESS_ACL	0x080
ACCESS_SUCCESS_MASK	0x0F0
ACCESS_FAIL_OPEN	0x100
ACCESS_FAIL_WRITE	0x200
ACCESS_FAIL_DELETE	0x400
ACCESS_FAIL_ACL	0x800
ACCESS_FAIL_MASK	0xF00
ACCESS_FAIL_SHIFT	0x004

The Access Permission APIs are:

4.12.5.1. NetAccessAdd

Only works when remoted to a downlevel system. Use Win32 (GetFileSecurity, SetFileSecurity) APIs for NT.

NetAccessAdd creates a new Access Control List (ACL) for a resource.

Privilege Level

Admin privilege is required to successfully execute **NetAccessAdd**.

NET_API_STATUS NET_API_FUNCTION
NetAccessAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information provided. Must be 1.

buf ___A pointer to a buffer containing the access information structure.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.5.2. NetAccessEnum

Only works when remoted to a downlevel system. Use Win32 (GetFileSecurity, SetFileSecurity) APIs for NT.

NetAccessEnum retrieves information about each access permision record.

NetAccessEnum is being phased out and is only available at levels 0 and 1 for Windows networking. It is recommended that NetAccessEnum not be used because it is unlikely to be supported in future releases.

Privilege Level

NetAccessEnum does not require admin privilege to execute but will only return records for entries for which the user has 'P' permission if the caller does not have admin privilege.

NET_API_STATUS NET_API_FUNCTION
NetAccessEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *basepath*,
IN DWORD *recursive*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

basepath ___A pointer to an UNICODE string that contains a base pathname for the resources. A NULL pointer or NULL string means no base path is to be used. The path can be specified as a universal naming convention (UNC) pathname.

recursive ___A flag to enables or disable recursive searching. If *recursive* is 0, NetAccessEnum returns entries only for the resource named as the base path by *basepath* and for the resources directly below that base path. If recursive is nonzero, NetAccessEnum returns entries for all access control lists (ACLs) that have *basepath* at the beginning of the resource name.

level ___Level of information required. Only 0 and 1 are valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing access search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.12.5.3. NetAccessGetInfo

Only works when remoted to a downlevel system. Use Win32 (GetFileSecurity, SetFileSecurity) APIs for NT.

NetAccessGetInfo retrieves the access control information for a specific resource.

Privilege Level

Admin privilege or P permission is required to successfully execute **NetAccessGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetAccessGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *resource*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

resource ___A pointer to an UNICODE string containing the name of the resource on which to return access control information.

level ___Level of information required. 0 and 1 are valid.

bufptr ___A pointer to the return information structure is returned in the address pointed to by bufptr.

4.12.5.4. NetAccessSetInfo

Only works when remoted to a downlevel system. Use Win32 (GetFileSecurity, SetFileSecurity) APIs for NT.

NetAccessSetInfo changes the access control list for a resource.

Privilege Level

Admin privilege or P permission for the resource is required to successfully execute **NetAccessSetInfo.**

NET_API_STATUS NET_API_FUNCTION
NetAccessSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *resource*,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

resource ___A pointer to an UNICODE string containing the name of the resource for which the access information shoule be changed.

level ___Level of information to set. Only 1 and 1002 are valid.

buf ___A pointer to a buffer containing the access entry information.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.12.5.5. NetAccessDel

Only works when remoted to a downlevel system. Use Win32 (GetFileSecurity, SetFileSecurity) APIs for NT.

NetAccessDel deletes the access control list for a particular resource.

Privilege Level

Admin privilege or P permission for the resource is required to successfully execute **NetAccessDel**.

NET_API_STATUS NET_API_FUNCTION
NetAccessDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *resource*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

resource ___A pointer to an UNICODE string containing the name of the resource for which to remove the access control entry.

4.12.5.6. NetAccessGetUserPerms

Only works when remoted to a downlevel system. Use Win32 (GetFileSecurity, SetFileSecurity) APIs for NT.

NetAccessGetUserPerms returns a specified user's or group's access permissions for a particular resource..

Privilege Level

Admin privilege or P permission for the resource is required to successfully execute **NetAccessGetUserPerms**.except when users request their own access permissions to a resource. In this case no special privilege is required.

NET_API_STATUS NET_API_FUNCTION
NetAccessGetUserPerms (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *UGname*,
IN LPWSTR *resource*,
OUT LPDWORD *Perms*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

UGname ___A pointer to an UNICODE string containing the name of the user or group to query permissions for.

resource ___A pointer to an UNICODE string containing the resource for which to query user permissions.

Perms ___Points to a DWORD in which the user permissions for the specified resource are returned.

4.12.5.7. NetAccessCheck

Only works when remoted to a downlevel system. Use Win32 (AccessCheck) API for NT.

NetAccessCheck verifies that a user has permission to perform a specified operation on a particular resource.

Privilege Level

Admin privilege is required to successfully execute **NetAccessCheck**.

NET_API_STATUS NET_API_FUNCTION
NetAccessCheck (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *username*,
IN LPWSTR *resource*,
IN DWORD *operation*,
OUT LPDWORD *result*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

username ___A pointer to an UNICODE string containing the name of the user to check for access to the resource.

resource ___A pointer to an UNICODE string containing the resource for which to check user permissions.

operation ___Points to a DWORD specifying the type of access operation requested.

ACCESS_READ	0x01	Permission to read data from a resource and, by default, execute the resource.
ACCESS_WRITE	0x02	Permission to write data to the resource.
ACCESS_CREATE	0x04	Permission to create an instance of the resource (for example, a file); data can be written to the resource when creating it.
ACCESS_EXEC	0x08	Permission to execute the resource.
ACCESS_DELETE	0x10	Permission to delete the resource.
ACCESS_ATRIB	0x20	Permission to modify the resource's attributes (for example, the date and time a file was last modified).
ACCESS_PERM	0x40	Permission to modify the permissions (read, write, create, execute, and delete) assigned to a resource for a user, group, or application.
ACCESS_ALL	0x7F	Permission to read, write, create, execute, or delete a resource, or to modify attributes or permissions.in which the user permissions for the specified resource are returned.

result ___Points to a DWORD in which the result of the access check is returned.

4.13. Domain APIs

Domain API functions retrieve information about a domain. They require that the workstation service be started.

The Domain APIs are:

4.13.1. NetGetDCName

NetGetDCName returns the name of the Primary Domain Controller for the specified domain.

Privilege Level

No special privilege is required to successfully execute **NetGetDCName**.

NET_API_STATUS NET_API_FUNCTION
NetGetDCName (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *domainname*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

domainname ___A pointer to an UNICODE string containing the name of the domain. A NULL pointer or string indicates that the name of the domain controller for the primary domain is to be returned.

bufptr ___A pointer to the buffer containing the domain controller name is returned in the address pointed to by bufptr.

4.14. Transport APIs

Transport APIs handle binding and unbinding of transports to/from the server and redirector, and also ennumerate the transports used by a component. The ServerTransport APIs deal with transports managed by the server, and the WkstaTransport APIs deal with transports managed by the redirector.

NetServerTransportAdd allows the user to bind the transport to the server.
NetServerTransportDel allows the user to unbind the server from the transport.
NetServerTransportEnum ennumerates the transports that are managed by the server.

NetWkstaTransportAdd/Del/Enum perform equivalent operations for the workstation.

ServerTransport APIs are available at one information level.

```
typedef struct _SERVER_TRANSPORT_INFO_0 {
        DWORD           svti0_number_of_vcs;
        LPWSTR          svti0_transport_name;
        LPBYTE          svti0_transport_address;
        DWORD           svti0_transportaddresslength;
        LPWSTR          svti0_networkaddress;
} SERVER_TRANSPORT_INFO_0, *PSERVER_TRANSPORT_INFO_0,
*LPSERVER_TRANSPORT_INFO_0;
```

WkstaTransport APIs are available at one information level.

```
typedef struct _WKSTA_TRANSPORT_INFO_0 {
        DWORD           wkti0_quality_of_service;
        DWORD           wkti0_number_of_vcs;
        LPWSTR          wkti0_transport_name;
        LPWSTR          wkti0_transport_address;
} WKSTA_TRANSPORT_INFO_0, *PWKSTA_TRANSPORT_INFO_0,
*LPWKSTA_TRANSPORT_INFO_0;
```

The APIs are:

4.14.1. NetServerTransportAdd

NetServerTransportAdd binds the server to the transport.

Privilege Level

Admin privilege is required to successfully execute **NetServerTransportAdd**.

NET_API_STATUS NET_API_FUNCTION
NetServerTransportAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. Only 0 is valid.

buf ___A pointer to a buffer containing the server transport information structure.

4.14.2. NetServerTransportDel

NetServerTransportDel unbinds the transport from the server.

Privilege Level

Admin privilege is required to successfully execute **NetServerTransportDel**.

NET_API_STATUS NET_API_FUNCTION
NetServerTransportDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *transportname*
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

transportname ___A pointer to an UNICODE string containing the name of the transport from which to unbind.

4.14.3. NetServerTransportEnum

NetServerTransportEnum supplies information about transports that are managed by the server.

Privilege Level

No special privilege is required to successfully execute **NetServerTransportEnum**.

NET_API_STATUS NET_API_FUNCTION
NetServerTransportEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. Only 0 is valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing server transport search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.14.4. NetWkstaTransportAdd

NetWkstaTransportAdd binds the redirector to the transport.

Privilege Level

Admin privilege is required to successfully execute **NetWkstaTransportAdd**.

NET_API_STATUS NET_API_FUNCTION
NetWkstaTransportAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*
);
Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. Only 0 is valid.

buf ___A pointer to a buffer containing the server transport information structure.

4.14.5. NetWkstaTransportDel

NetWkstaTransportDel unbinds the transport from the redirector.

Privilege Level

Admin privilege is required to successfully execute **NetWkstaTransportDel**.

NET_API_STATUS NET_API_FUNCTION
NetWkstaTransportDel (
IN LPWSTR *servername* **OPTIONAL,**
IN LPWSTR *transportname***,**
IN DWORD *ucond*
);
Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

transportname ___A pointer to an UNICODE string containing the name of the transport from which to unbind.

ucond ___Force level to delete connections on the transport binding.

4.14.6. NetWkstaTransportEnum

NetWkstaTransportEnum supplies information about transports that are managed by the redirector.

Privilege Level

No special privilege is required to successfully execute **NetWkstaTransportEnum**.

NET_API_STATUS NET_API_FUNCTION
NetWkstaTransportEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to an UNICODE string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. Only 0 is valid.

bufptr ___On return a pointer to the return information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

entriesread ___On return the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing workstation transport search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored..

4.15. Alert APIs

Alert API functions notify network service programs and applications of network events. An event is a particular instance of a process or state of hardware as defined by an application. The Alert API functions allow applications to indicate when predefined events occur.

NetAlertRaise is used to indicate that an event has occurred.

The fixed-length header contains the standard alert data structure. The **STD_ALERT** data structure has the following format:

```
typedef struct _STD_ALERT {
        DWORD           alrt_timestamp;
        WCHAR           alrt_eventname[EVLEN+1];
        WCHAR           alrt_servicename[SNLEN+1];
} STD_ALERT, *PSTD_ALERT, *LPSTD_ALERT;
```

The **ADMIN_OTHER_INFO** data structure has this format:

```
typedef struct _ADMIN_OTHER_INFO {
        DWORD           alrtad_errcode;
        DWORD           alrtad_numstrings;
} ADMIN_OTHER_INFO, *PADMIN_OTHER_INFO, *LPADMIN_OTHER_INFO;

/* Followed by consecutive UNICODE strings; the count
   is in the alrtad_numstrings element. */

        WCHAR           mergestrings[];
```

The **ERRLOG_OTHER_INFO** data structure has this format:

```
typedef struct _ERRLOG_OTHER_INFO {
        DWORD           alrter_errcode;
        DWORD           alrter_offset;
} ERRLOG_OTHER_INFO, *PERRLOG_OTHER_INFO,
*LPERRLOG_OTHER_INFO;
```

The **PRINT_OTHER_INFO** data structure has this format:

```
typedef struct _PRINT_OTHER_INFO {
        DWORD           alrtpr_jobid;
        DWORD           alrtpr_status;
        DWORD           alrtpr_submitted;
        DWORD           alrtpr_size;
} PRINT_OTHER_INFO, *PPRINT_OTHER_INFO, *LPPRINT_OTHER_INFO;
/* Followed by consecutive UNICODE strings. */

        WCHAR           computername[];
        WCHAR           username[];
        WCHAR           queuename[];
        WCHAR           destname[];
        WCHAR           status_string[];
```

The **USER_OTHER_INFO** data structure has this format:

```
typedef struct _USER_OTHER_INFO {
        DWORD           alrtus_errcode;
        DWORD           alrtus_numstrings;
} USER_OTHER_INFO, *PUSER_OTHER_INFO, *LPUSER_OTHER_INFO;
/*
 * Followed by a number of consecutive UNICODE strings;
 * the count is in the alrtus_numstrings element. */
 */
        WCHAR           mergestrings[];

/* Further followed by two more consecutive UNICODE strings. */
        WCHAR           username[];
        WCHAR           computername[];
```

The APIs are:

4.15.1. NetAlertRaise

NetAlertRaise notifies all registered clients that a particular event occurred.

Privilege Level

No special privilege level is required to successfully execute **NetAlertRaise**.

NET_API_STATUS NET_API_FUNCTION
NetAlertRaise (
IN LPWSTR *event*,
IN LPBYTE *buffer*,
IN DWORD *numbytes*
);

Parameters:

event ___Points to an UNICODE string that specifies which type of alert to raise.

buffer ___Points to the data to be sent to the clients listening for this alert. The data should consist of the STD_ALERT data structure followed by any additional alert data.

numbytes ___Specifies the size of the buffer in bytes.

4.15.2. NetAlertRaiseEx

NetAlertRaiseEx simplifies the raising of an admin alert.

Privilege Level

No special privilege level is required to successfully execute **NetAlertRaiseEx**.

NET_API_STATUS NET_API_FUNCTION
NetAlertRaiseEx (
IN LPWSTR *event*,
IN LPVOID *VariableInfo*,
IN DWORD *VariableInfoSize,*
IN LPWSTR *ServiceName*
);

Parameters:

event ___Points to an UNICODE string that specifies which type of alert to raise.

VariableInfo __Information to put into the admin alert.

VariableInfoSize __Number of bytes of variable information.

ServiceName __Name of the service raising the admin alert.

4.16. Error Logging APIs

Only use these for remoting to downlevel. For NT use the EventLog API (ReadEventlog, ReportEvent, ...)

Windows NT uses an integrated Eventlogging mechanism for reporting both errors and audits. The NetError and NetAudit APIs are provided to access downlevel LANMan logs. They will report ERROR_NOT_SUPPORTED if called to an Windows NT system.

The error log is a file that stores error messages (in binary format). It contains information about LAN Manager software internal errors, MS OS/2 and MS-DOS internal errors, and network service errors.

NetErrorLogRead reads entries from the error log; and **NetErrorLogClear** clears the error log and, optionally, saves the entries in a backup file.

NetErrorLogRead uses the **ERROR_LOG** data structure to read entries from and write entries to the error log. An entry consists of a fixed-length data structure. The data structure can be followed by UNICODE strings (**el_text**) that describe the error message, and a block of raw data (**el_data**) related to the cause of the error. Because of the variable lengths and structures of the **el_data** and **el_text** portions of the entry, only the fixed-length data structure is defined in the **ERROR_LOG** data structure.

The error log entry has the following format:

```
typedef struct _ERROR_LOG {
        DWORD           el_len;
        DWORD           el_reserved;
        DWORD           el_time;
        DWORD           el_error;
        LPWSTR          el_name
        LPWSTR          el_text;
        LPBYTE          el_data;
```

```
        DWORD           el_data_size;
        DWORD           el_nstrings;
} ERROR_LOG, *PERROR_LOG, *LPERROR_LOG;

/*
 *  Variable-length raw data specific to the error. el_len will be preceded by pad bytes
 *   as required for proper alignment.  There may also be pad byes after the end of the
 *   fixed length structure, before el_data.  Use the pointer in the fixed length structure
 *   to reference this data.
 */
        BYTE            el_data[];    /* Raw data      */
        DWORD           el_len2;
```

The error log handle is defined as follows:

```
typedef struct _HLOG {
        DWORD           time;
        DWORD           last_flags;
        DWORD           offset;
        DWORD           rec_offset;
} HLOG, *PHLOG, *LPHLOG;
```

The error log APIs are:

4.16.1. NetErrorLogClear

NetErrorLogClear clears the error log and optionally saves the entries in a backup file.

Privilege Level

Admin privilege is required to successfully execute **NetErrorLogClear** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetErrorLogClear (
IN LPWSTR *server*,
IN LPWSTR *backupfile* **OPTIONAL**,
IN LPBYTE *reserved* **OPTIONAL**
);

Parameters:

server ___Points to an UNICODE string that contains the name of the server on which to execute NetErrorLogClear. A NULL pointer or NULL string specifies the local computer.

backupfile ___Points to an UNICODE string that assigns a name for an optional backup file. The calling application must have write permission for the path specified by backupfile. A NULL pointer indicates not to back up the error log.

reserved ___Reserved. Must be NULL.

4.16.2. NetErrorLogRead

NetErrorLogRead reads from the specified error log.

Privilege Level

No special privilege level is required to successfully execute **NetErrorLogRead**.

NET_API_STATUS NET_API_FUNCTION
NetErrorLogRead (
IN LPWSTR *server*,
IN LPWSTR *reserved1* **OPTIONAL**,
IN LPHLOG *errloghandle*,
IN DWORD *offset*,
IN LPDWORD *reserved2* **OPTIONAL**,
IN DWORD *reserved3*,
IN DWORD *offsetflag*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *bytesread*,
OUT LPDWORD *totalbytes*
);

Parameters:

server ___Points to an UNICODE string that contains the name of the server on which to execute NetErrorLogClear. A NULL pointer or NULL string specifies the local computer.

reserved1 ___Reserved. Must be NULL.

errloghandle ___Points to the error log handle. An application calling NetErrorLogRead for the first time must initialize the 128-bit error log handle.

offset ___Specifies the record offset at which to begin reading. This parameter is ignored unless bit 1 of offsetflag is set. If used, offset is taken as an offset of the record number (not bytes) at which to begin reading. Note that the record offset parameter is zero-based from both directions, depending upon the direction it is read.

reserved2 ___Reserved. Must be NULL.

reserved3 ___Reserved. Must be zero.

offsetflag ___Specifies the open flags, bitmapped as follows:

Bit(s)	Value
4.16.2.1.	If 0, the log is read forward. If 1, the log is read backward and records are returned in reverse chronological order.
4.16.2.2.	If 0, the read proceeds normally (sequentially). If 1, the read proceeds from

the nth record from the start of the log, where n is the offset parameter.

4.16.2.3. Reserved; must be 0.

bufptr ___On return a pointer to the return information buffer is returned in the address pointed to by bufptr.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

bytesread ___On return the actual number of bytes read into the buffer is located in the DWORD pointed to by bytesread.

totalbytes ___On return the total bytes available to be read from the log is located in the DWORD pointed to by totalbytes.

4.17. Configuration APIs

These are only for downlevel support. Use the Registry APIs to retrieve configuration information on NT.

Configuration API functions retrieve network configuration information.

Configuration APIs are available at one information level.

```
typedef struct _CONFIG_INFO_0 {
        LPWSTR          cfgi0_key;
        LPWSTR          cfgi0_data;
} CONFIG_INFO_0, *PCONFIG_INFO_0, *LPCONFIG_INFO_0;
```

The Configuration APIs are:

4.17.1. NetConfigGet

NetConfigGet retrieves the value of a single specified entry for a particular component on the local computer or on a remote server.

Privilege Level

Admin privilege or accounts, comm, print, or server operator privilege is required to successfully execute **NetConfigGet** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetConfigGet (
IN LPWSTR *server*,
IN LPWSTR *component*,
IN LPWSTR *parameter*,
OUT LPBYTE * *bufptr*
);

Parameters:

server ___Points to an UNICODE string that contains the name of the server on which to execute NetConfigGet. A NULL pointer or NULL string specifies the local computer.

component ___Points to an UNICODE string that specifies which configuration component to search.

parameter ___Points to an UNICODE string that specifies the entry for the component specified whose value is to be returned.

bufptr ___On return a pointer to the returned information is returned in the address pointed to by bufptr.

4.17.2. NetConfigGetAll

NetConfigGetAll retrieves all the configuration information for a given component on a local or a remote computer.

Privilege Level

Admin privilege or account, comm, print, or server operator privilege is required to successfully execute **NetConfigGetAll** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetConfigGetAll (
IN LPWSTR *server*,
IN LPWSTR *component*,
OUT LPBYTE * *bufptr*
);

Parameters:

server ___Points to an UNICODE string that contains the name of the server on which to execute NetConfigGet. A NULL pointer or NULL string specifies the local computer.

component ___Points to an UNICODE string that specifies which configuration component to search.

bufptr ___On return a pointer to the returned information is returned in the address pointed to by bufptr.

4.17.3. NetConfigSet

NetConfigSet sets the value of a single specified entry for a particular component on the local computer or on a remote server.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetConfigSet** on a remote server.

NET_API_STATUS NET_API_FUNCTION
NetConfigSet (
IN LPWSTR *server*,
IN LPWSTR *reserved1* **OPTIONAL**,
IN LPWSTR *component*,
IN DWORD *level*,
IN DWORD *reserved2*,
IN LPBYTE *buf*,
IN DWORD *reserved3*
);

Parameters:

server ___Points to an UNICODE string that contains the name of the server on which to execute NetConfigSet. A NULL pointer or NULL string specifies the local computer.

reserved1 ___Reserved. Must be NULL.

component ___Points to an UNICODE string that specifies which configuration component to set.

level ___Infolevel of request. Must be zero.

reserved2 ___Reserved. Must be zero.

buf ___Buffer containing a CONFIG_INFO_0 structures.

reserved3 ___Reserved. Must be zero.

4.18. Statistics APIs

LAN Manager accumulates a set of operating statistics for workstations and servers from the time that the Workstation or Server service is started. **NetStatisticsGet** is called to get those statistics. The Statistics API function, **NetStatisticsGet**, retrieves the operating statistics for workstations and servers. Since NT and downlevel workstations collect a different set of statistics, the caller must know whether the server is NT or downlevel (which can be discovered via the NetServerGetInfo API) and interpret the returned buffer accordingly.

The way lmstats.h is defined now, you can't define both an uplevel and downlevel structure without standing on your head (doing undef's etc). We need to fix this so both structures have unique names.

NetStatisticsGet returns a **STAT_WORKSTATION_0** data structure when workstation statistics are requested; it returns a **STAT_SERVER_0** data structure when server statistics are requested.

The downlevel **STAT_WORKSTATION_0** data structure has this format:

```
typedef struct _STAT_WORKSTATION_0 {
        DWORD           stw0_start;
        DWORD           stw0_numNCB_r;
        DWORD           stw0_numNCB_s;
        DWORD           stw0_numNCB_a;
        DWORD           stw0_fiNCB_r;
        DWORD           stw0_fiNCB_s;
        DWORD           stw0_fiNCB_a;
        DWORD           stw0_fcNCB_r;
        DWORD           stw0_fcNCB_s;
        DWORD           stw0_fcNCB_a;
        DWORD           stw0_sesstart;
        DWORD           stw0_sessfailcon;
        DWORD           stw0_sessbroke;
        DWORD           stw0_uses;
        DWORD           stw0_usefail;
        DWORD           stw0_autorec;
        DWORD           stw0_bytessent_r_hi;
        DWORD           stw0_bytessent_r_lo;
        DWORD           stw0_bytesrcvd_r_hi;
        DWORD           stw0_bytesrcvd_r_lo;
        DWORD           stw0_bytessent_s_hi;
        DWORD           stw0_bytessent_s_lo;
        DWORD           stw0_bytesrcvd_s_hi;
        DWORD           stw0_bytesrcvd_s_lo;
        DWORD           stw0_bytessent_a_hi;
        DWORD           stw0_bytessent_a_lo;
        DWORD           stw0_bytesrcvd_a_hi;
        DWORD           stw0_bytesrcvd_a_lo;
        DWORD           stw0_reqbufneed;
        DWORD           stw0_bigbufneed;
} STAT_WORKSTATION_0, *PSTAT_WORKSTATION_0,
*LPSTAT_WORKSTATION_0;
```

The NT **STAT_WORKSTATION_0** data structure has this format:

```
typedef struct _STAT_WORKSTATION_0 {
        LARGE_INTEGER   StatisticsStartTime;
        LARGE_INTEGER   BytesReceived;
        LARGE_INTEGER   SmbsReceived;
        LARGE_INTEGER   PagingReadBytesRequested;
        LARGE_INTEGER   NonPagingReadBytesRequested;
        LARGE_INTEGER   CacheReadBytesRequested;
        LARGE_INTEGER   NetworkReadBytesRequested;
        LARGE_INTEGER   BytesTransmitted;
        LARGE_INTEGER   SmbsTransmitted;
        LARGE_INTEGER   PagingWriteBytesRequested;
        LARGE_INTEGER   NonPagingWriteBytesRequested;
        LARGE_INTEGER   CacheWriteBytesRequested;
        LARGE_INTEGER   NetworkWriteBytesRequested;
        DWORD           InitiallyFailedOperations;
        DWORD           FailedCompletionOperations;
        DWORD           ReadOperations;
        DWORD           RandomReadOperations;
        DWORD           ReadSmbs;
        DWORD           LargeReadSmbs;
        DWORD           SmallReadSmbs;
        DWORD           WriteOperations;
        DWORD           RandomWriteOperations;
        DWORD           WriteSmbs;
        DWORD           LargeWriteSmbs;
        DWORD           SmallWriteSmbs;
        DWORD           RawReadsDenied;
        DWORD           RawWritesDenied;
        DWORD           NetworkErrors;
        //  Connection/Session counts
        DWORD           Sessions;
        DWORD           FailedSessions;
        DWORD           Reconnects;
        DWORD           CoreConnects;
        DWORD           Lanman20Connects;
        DWORD           Lanman21Connects;
        DWORD           LanmanNtConnects;
        DWORD           ServerDisconnects;
        DWORD           HungSessions;
        DWORD           UseCount;
        DWORD           FailedUseCount;
        //
        //  Queue Lengths (updates protected by RdrMpxTableSpinLock NOT
        //  RdrStatisticsSpinlock)
        //
        DWORD           CurrentCommands;
} STAT_WORKSTATION_0, *PSTAT_WORKSTATION_0,
*LPSTAT_WORKSTATION_0;
```

The **STAT_SERVER_0** data structure has this format:

```
typedef struct _STAT_SERVER_0 {
        DWORD           sts0_start;
        DWORD           sts0_fopens;
        DWORD           sts0_devopens;
        DWORD           sts0_jobsqueued;
        DWORD           sts0_sopens;
        DWORD           sts0_stimedout;
        DWORD           sts0_serrorout;
        DWORD           sts0_pwerrors;
        DWORD           sts0_permerrors;
        DWORD           sts0_syserrors;
        DWORD           sts0_bytessent_low;
        DWORD           sts0_bytessent_high;
        DWORD           sts0_bytesrcvd_low;
        DWORD           sts0_bytesrcvd_high;
        DWORD           sts0_avresponse;
        DWORD           sts0_reqbufneed;
        DWORD           sts0_bigbufneed;
} STAT_SERVER_0, *PSTAT_SERVER_0, *LPSTAT_SERVER_0;
```

4.18.1. NetStatisticsGet

NetStatisticsGet retrieves, and optionally clears, operating statistics for a service. Currently, only the Workstation and Server services are supported.

Privilege Level

Admin privilege or server operator privilege is required to successfully execute **NetStatisticsGet** on a remote server.

```
NET_API_STATUS NET_API_FUNCTION
NetStatisticsGet (
        IN LPWSTR server,
        IN LPWSTR service,
        IN DWORD level,
        IN DWORD options,
        OUT LPBYTE * bufptr
        );
```

Parameters:

server —Points to an UNICODE string that contains the name of the server on which to execute NetConfigGet. A NULL pointer or NULL string specifies the local computer.

service —Points to an UNICODE string that contains the name of the service about which to get the statistics. Only the values SERVER and WORKSTATION are currently allowed.

level —Specifies the level of detail requested; must be 0.

options —Specifies the options flags.

Bit(s)	Meaning
4.18.1.1.	Clear statistics.
4.18.1.2.	Reserved; must be 0.

bufptr ___On return a pointer to the returned information is returned in the address pointed to by bufptr.

4.19. Auditing APIs

These are for downlevel only. NT automatically records audits on failed accesses when AccessCheckAndAudit is used. To read audits, use the Eventlog ReadEventlog API.

Windows NT uses an integrated Eventlogging mechanism for reporting both errors and audits. The NetError and NetAudit APIs are provided to access downlevel LANMan logs. They will report ERROR_NOT_SUPPORTED if called to an Windows NT system.

Auditing API functions control the audit log. Auditing API functions monitor operations on the specified server. If auditing is enabled, each monitored operation generates an audit entry. For example, when a user establishes a connection to the server, a single audit entry is generated.

Audit entries are stored in a binary file called an audit trail or audit log. All Auditing API functions perform their operations on this file. LAN Manager defines many types of audit entries.

NetAuditRead reads the audit log. **NetAuditClear** clears the audit log.

Data Structures

All audit entries include a fixed-length header used in conjunction with variable-length data specific to the entry type. Because of the variable lengths and structures of the **ae_data** element of the audit entry (it is possible for **ae_data** to be zero bytes), only the fixed header is defined in the **audit_entry** data structure.

The variable-length portion of the audit entry can contain an offset to a variable-length UNICODE string. The offset values are DWORDs. To determine the value of the pointer to this string, add the offset value to the address of the **ae_data** data structure.

The following example illustrates this procedure. Assume that **pAE** points to a buffer that contains a complete audit entry and that the **ae_type** element of the **audit_entry** data structure contains the value AE_CONNSTOP, which specifies the predefined **AE_CONNSTOP** data structure. To point the variable **pszComputerName** to the UNICODE string that contains the name of the client whose connection was stopped, an application would perform the following algorithm:

```
PAUDIT_ENTRY    pAE;                /* Fixed part of audit entry */
LPAE_CONNSTOP   pAEvar;             /* Variable-length structure */
LPWSTR          pszComputerName;    /* Pointer to var-length string*/

/* Calculate the offset to the variable-length structure. */
```

```
    pAEvar = (_LPAE_CONNSTOP) (((LPBYTE) pAE) + pAE->ae_data_offset);

    /* Calculate the offset to the computername. */
    pszComputerName = ((LPBYTE) pAEvar) + pAEvar->ae_cp_compname;
```

Fixed-Length Header

The **audit_entry** data structure has this format:

```
typedef struct _AUDIT_ENTRY {
        DWORD           ae_len;
        DWORD           ae_reserved;
        DWORD           ae_time;
        DWORD           ae_type;
        DWORD           ae_data_offset;  /* Offset from beginning
                                            address of audit_entry */

        DWORD           ae_data_size;
} AUDIT_ENTRY, *PAUDIT_ENTRY, *LPAUDIT_ENTRY;

/* Variable-length data specific to type of audit entry.  There may be pad bytes after
    ae_data_size for alignment purposes.
*/

        BYTE            ae_data[];

/* Terminating length indicator.  Preceded by pad bytes for proper alignment.*/

        DWORD           ae_len2;
```

where

ae_len and *ae_len2* Specify the length of the audit entry. Both have the same value. The ae_len element is included at the beginning and at the end of the audit entry to enable both backward and forward scanning of the log.

ae_reserved Reserved.

ae_time Specifies when the audit entry was generated. The value is stored as the number of seconds elapsed since 00:00:00, January 1, 1970.

ae_type Specifies the type of audit entry. Type values from 0x0000 through 0x07FF are reserved. OEMs and other applications programmers can reserve values from 0x0800 through 0xFFFF.

ae_data_offset Specifies the byte offset from the beginning of the audit entry to the beginning of the variable-length portion (ae_data) of the audit entry.

ae_data Specifies the variable-length portion of the audit entry; it differs depending on the type of entry specified by ae_type. The information begins at ae_data_offset bytes from the top of the audit entry. For information about the structure of each entry type defined by LAN Manager, see the following section.

Variable-Length Data

The following data structuresare specific to the audit entry type The structures follow the **audit_entry** header, but they are not necessarily contiguous.

The **_AE_SRVSTATUS** data structure is associated with an audit entry of type AE_SRVSTATUS.

```
typedef struct _AE_SRVSTATUS {
        DWORD           ae_sv_status;
}AE_SRVSTATUS , * PAE_SRVSTATUS ;
```

The **_AE_SESSLOGON** data structure is associated with an audit entry of type AE_SESSLOGON.

```
typedef struct _AE_SESSLOGON {
        DWORD           ae_so_compname;
        DWORD           ae_so_username;
        DWORD           ae_so_privilege;
}AE_SESSLOGON , * PAE_SESSLOGON ;
```

The **_AE_SESSLOGOFF** data structure is associated with an audit entry of type AE_SESSLOGOFF.

```
typedef struct AE_SESSLOGOFF {
        DWORD           ae_sf_compname;
        DWORD           ae_sf_username;
        DWORD           ae_sf_reason;
}AE_SESSLOGOFF , * PAE_SESSLOGOFF;
```

The **_AE_SESSPWERR** data structure is associated with an audit entry of type AE_SESSPWERR.

```
typedef struct _AE_SESSPWERR {
        DWORD           ae_sp_compname;
        DWORD           ae_sp_username;
}AE_SESSPWERR , * PAE_SESSPWERR ;
```

The **_AE_CONNSTART** data structure is associated with an audit entry of type AE_CONNSTART.

```
typedef struct _AE_CONNSTART {
        DWORD           ae_ct_compname;
        DWORD           ae_ct_username;
        DWORD           ae_ct_netname;
        DWORD           ae_ct_connid;
}AE_CONNSTART , * PAE_CONNSTART ;
```

The **_AE_CONNSTOP** data structure is associated with an audit entry of type AE_CONNSTOP.

```
typedef struct _AE_CONNSTOP {
        DWORD           ae_cp_compname;
        DWORD           ae_cp_username;
        DWORD           ae_cp_netname;
        DWORD           ae_cp_connid;
        DWORD           ae_cp_reason;
}AE_CONNSTOP , * PAE_CONNSTOP ;
```

The **_AE_CONNREJ** data structure is associated with an audit entry of type AE_CONNREJ.

```
typedef struct _AE_CONNREJ {
        DWORD           ae_cr_compname;
        DWORD           ae_cr_username;
        DWORD           ae_cr_netname;
        DWORD           ae_cr_reason;
}AE_CONNREJ , * PAE_CONNREJ ;
```

The **_AE_RESACCESS** data structure is associated with an audit entry of type AE_RESACCESS.

```
typedef struct _AE_RESACCESS {
        DWORD           ae_ra_compname;
        DWORD           ae_ra_username;
        DWORD           ae_ra_resname;
        DWORD           ae_ra_operation;
        DWORD           ae_ra_returncode;
        DWORD           ae_ra_restype;
        DWORD           ae_ra_fileid;
}AE_RESACCESS , * PAE_RESACCESS ;
```

The **_AD_RESACCESSREJ** data structure is associated with an audit entry of type AE_RESACCESSREJ.

```
typedef struct _AE_RESACCESSREJ {
        DWORD           ae_rr_compname;
        DWORD           ae_rr_username;
        DWORD           ae_rr_resname;
        DWORD           ae_rr_operation;
}AE_RESACCESSREJ , * PAE_RESACCESSREJ ;
```

The **_AD_CLOSEFILE** data structure is associated with an audit entry of type AE_CLOSEFILE.

```
typedef struct _AE_CLOSEFILE {
        DWORD           ae_cf_compname;
        DWORD           ae_cf_username;
        DWORD           ae_cf_resname;
        DWORD           ae_cf_fileid;
        DWORD           ae_cf_duration
        DWORD           ae_cf_reason;
}AE_CLOSEFILE , * PAE_CLOSEFILE ;
```

The **_AE_SERVICESTAT** data structure is associated with an audit entry of type AE_SERVICESTAT.

```
typedef struct _AE_SERVICESTAT {
        DWORD           ae_ss_compname;
        DWORD           ae_ss_username;
        DWORD           ae_ss_svcname;
        DWORD           ae_ss_status;
        DWORD           ae_ss_code;
        DWORD           ae_ss_text;
        DWORD           ae_ss_returnval;
}AE_SERVICESTAT , * PAE_SERVICESTAT ;
```

The **_AE_ACLMOD** data structure is associated with audit entries of type AE_ACLMOD and AE_ACLMODFAIL.

```
struct _AE_ACLMOD {
        DWORD           ae_am_compname;
        DWORD           ae_am_username;
        DWORD           ae_am_resname;
        DWORD           ae_am_action;
        DWORD           ae_am_datalen;
}AE_ACLMOD, * PAE_ACLMOD;
```

The **_AE_UASMOD** data structure is associated with an audit entry of type AE_UASMOD.

```
typedef struct _AE_UASMOD {
        DWORD           ae_um_compname;
        DWORD           ae_um_username;
        DWORD           ae_um_resname;
        DWORD           ae_um_rectype;
        DWORD           ae_um_action;
        DWORD           ae_um_datalen;
}AE_UASMOD , * PAE_UASMOD ;
```

The **_AE_NETLOGON** data structure is associated with an audit entry of type AE_NETLOGON.

```
typedef struct _AE_NETLOGON {
        DWORD           ae_no_compname;
        DWORD           ae_no_username;
        DWORD           ae_no_privilege;
        DWORD           ae_no_authflags;
}AE_NETLOGON , * PAE_NETLOGON ;
```

The **_AE_NETLOGOFF** data structure is associated with an audit entry of type AE_NETLOGOFF.

```
typedef struct _AE_NETLOGOFF {
        DWORD           ae_nf_compname;
        DWORD           ae_nf_username;
        DWORD           ae_nf_reserved1;
        DWORD           ae_nf_reserved2;
}AE_NETLOGOFF , * PAE_NETLOGOFF ;
```

The **_AE_ACCLIM** data structure is associated with an audit entry of type AE_ACCLIMITEXCD.

```
typedef struct _AE_ACCLIM {
        DWORD           ae_al_compname;
        DWORD           ae_al_username;
        DWORD           ae_al_resname;
        DWORD           ae_al_limit;
}AE_ACCLIM , * PAE_ACCLIM ;
```

The **_AE_LOCKOUT** data structure is associated with an audit entry of type AE_LOCKOUT.

```
typedef struct AE_LOCKOUT {
        DWORD           ae_lk_compname;
        DWORD           ae_lk_username;
        DWORD           ae_lk_action;
        DWORD           ae_lk_bad_pw_count;
}AE_LOCKOUT , PAE_LOCKOUT ;
```

The **_AE_GENERIC** data structure is associated with an audit entry of type AE_GENERIC.

```
typedef struct _AE_GENERIC {
        DWORD           ae_ge_msgfile;
        DWORD           ae_ge_msgnum;
        DWORD           ae_ge_params;
        DWORD           ae_ge_param1;
        DWORD           ae_ge_param2;
        DWORD           ae_ge_param3;
        DWORD           ae_ge_param4;
        DWORD           ae_ge_param5;
        DWORD           ae_ge_param6;
        DWORD           ae_ge_param7;
        DWORD           ae_ge_param8;
        DWORD           ae_ge_param9;
}AE_GENERIC , * PAE_GENERIC ;
```

The audit log handle is defined as follows:

```
typedef struct _HLOG {
        DWORD           time;
        DWORD           last_flags;
        DWORD           offset;
        DWORD           rec_offset;
} HLOG, *PHLOG, *LPHLOG;
```

4.19.1. NetAuditClear

NetAuditClear clears the audit log on a server and, optionally, saves the entries in a backup file.

Privilege Level

Admin privilege is required to successfully execute **NetAuditClear** on a remote server or on a computer that has local security enabled.

NET_API_STATUS NET_API_FUNCTION
NetAuditClear (
IN LPWSTR *server*,
IN LPWSTR *backupfile* **OPTIONAL**,
IN LPWSTR *service* **OPTIONAL**
);

Parameters:

server __Points to an UNICODE string that contains the name of the server on which to execute NetAuditClear. A NULL pointer or NULL string specifies the local computer.

backupfile __Points to an UNICODE string that contains a name for the optional backup file. The calling application must have create and write permissions for the path specified by backupfile, and the path must already exist. A NULL pointer specifies not to back up the audit log.

service __Points to an UNICODE string that contains the name of the service that owns the desired audit log to which the operation is to be performed.

4.19.2. NetAuditRead

NetAuditRead reads from the audit log on a server.

Privilege Level

Admin privilege or accounts, comm, print, or server operator privilege is required to successfully execute **NetAuditRead** on a remote server or on a computer that has local security enabled.

NET_API_STATUS NET_API_FUNCTION
NetAuditRead (
IN LPWSTR *server*,
IN LPWSTR *service* **OPTIONAL,**
IN LPHLOG *auditloghandle*,
IN DWORD *offset*,
IN LPDWORD *reserved1* **OPTIONAL**,
IN DWORD *reserved2*,
IN DWORD *offsetflag*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *bytesread*,
OUT LPDWORD *totalavailable*
);

Parameters:

server __Points to an UNICODE string that contains the name of the server on which to execute NetAuditRead. A NULL pointer or NULL string specifies the local computer.

service __Points to an UNICODE string that contains the name of the service that owns the desired audit log to which the operation is to be performed.

auditloghandle ___Points to the handle for the audit log. An application calling NetAuditRead for the first time must initialize the audit log handle as follows. The most significant bit (MSB) is the leftmost bit; the least significant bit (LSB) is the rightmost bit. After the first call, each call to NetAuditRead must be given the value for the handle returned by the previous call.

	auditloghandle	**Value**
4.19.2.1.	(MSB) - 64	0
4.19.2.2.	- 0 (LSB)	1

offset ___Specifies the record offset at which to begin reading. This parameter is ignored unless bit 1 in the offsetflag parameter is set. If offsetflag bit 1 is set, offset is taken as an offset based on the record number (not bytes) at which the returned data should begin. Note that the record offset parameter is zero-based from both directions, depending upon the direction of the read. If reading backward, record 0 is the last record in the log. If reading forward, record 0 is the first record in the log.

reserved1 ___Reserved. Must be NULL.

reserved2 ___Reserved. Must be zero.

offsetflag ___Specifies the open flags, as follows:

Bit(s)	**Meaning**
4.19.2.3.	If 0, the log is read forward**.** If 1, the log is read backward and records are returned in the buffer in reverse chronological order (newest records first).
4.19.2.4.	If 0, reading proceeds sequentially. If 1, reading proceeds from the nth record from the start of the log, where n is the offset parameter.
4.19.2.5.	Reserved; must be 0**.**

bufptr ___On return a pointer to the returned information is returned in the address pointed to by bufptr. After a successful read operation, this buffer contains a sequence of audit entries with the accompanying variable-length data structures.

prefmaxlen ___Prefered maximum length of returned data (in 8-bit bytes).

bytesread ___On return the actual number of bytes read into the buffer is located in the DWORD pointed to by bytesread.

totalavailable ___On return the total bytes available to be read from the log is located in the DWORD pointed to by totalavailable.

4.19.3. NetAuditWrite

This looks like a local only API. If so, remove it since these are for downlevel support only.

NetAuditWrite writes an audit entry to the local audit log.

Privilege Level

No special privilege level is required to successfully execute **NetAuditWrite**.

NET_API_STATUS NET_API_FUNCTION
NetAuditWrite (
IN DWORD *type*,
IN LPBYTE *buf*,
IN DWORD *numbytes*
IN LPWSTR *service*,
IN LPBYTE *reserved*,
);

Parameters:

type ___Specifies the type of entry to write to the audit log.

buf ___Points to a buffer that contains the data structure associated with the specified audit entry.

numbytes ___Specifies the size (in bytes) of the buffer pointed to by buf.

service ___Points to an UNICODE string that contains the name of the service that owns the desired audit log to which the operation is to be performed.

reserved ___Reserved. Must be NULL.

4.20. Replicator APIs

The NT Replicator APIs control how the NT Replicator service updates selective directories from an export server to one or more clients. In addition to providing compatible LAN Manager 2.x functionality in a well-defined manner, this new set of LAN Manager APIs allow for specific API (operation) security checking.

The change from the file system based control on LAN Manager 2.x to Replicator API control on NT has the following implications:

- Applications can no longer delete a directory in the import path of a client to stop receiving updates from its master.
- Applications can no longer use the REPL.INI file in each replicated directory on a master to control the method of replication.
- Applications can no longer lock or unlock a directory on a master from being replicated by creating or deleting the USERLOCK.* file(s).
- Applications can no longer lock or unlock a directory on a client from receiving updates from its master by creating or deleting the USERLOCK.* file(s).

- Applications which depend on the LAN Manager 2.x behavior of ignoring locks for file integrity trees will need to be modified. (NT policy differs from LAN Manager 2.x policy; under NT the locks are always respected.)

Each of the intended operations listed above can be specified to the NT Replicator service through an appropriate API.

Any user or application which belongs to the admin or server operator group on a local or remote export server can modify the parameters which control the replication master.

There are three categories of Replicator APIs:

- Replicator Configuration APIs
- Replicator Export Directory APIs
- Replicator Import Directory APIs.

4.20.1. Replicator Configuration APIs

The configuration parameters of the Replicator service can be examined using **NetReplGetInfo**. They can be modified using **NetReplSetInfo**.

The Replicator service configuration parameters are:

role - Role of the Replicator service which is either REPL_ROLE_IMPORT (client), REPL_ROLE_EXPORT (master), or REPL_ROLE_BOTH. On a Advanced Server, any of these values are allowed. Under Windows/NT systems, only REPL_ROLE_IMPORT is allowed.

exportpath - Fully-qualified path name to the master tree in which directories are created and replicated from. This path must include the drive letter.

exportlist - List of machine and domain names to send update announcements to. If this list is not provided, then update announcements will be sent to the server's domain. The list entries are separated by semicolons; machine names in the list must not have leading backslashes

importpath - Fully-qualified path name to the client tree in which directories are created to receive the replicas of the master. This path must include the drive letter.

importlist - List of machines and domain names to receive updates from. If this list is not provided, then update announcements are received from the server's domain. The list entries are separated by semicolons; machine names in the list must not have leading backslashes.

logonusername - User account name belonging to the replicator group which the client logs on with to read files from its master(s). This field is ignored by the NT replicator, but is provided for possible future use.

interval - Time in minutes within which the master checks for changes in all the replicated directories. This field may be ignored by the NT replicator, but is provided for possible future use.

pulse - Time in number of **intervals** within which the master notifies its clients of the current replica version when no updates are necessary. This field may be ignored by the NT replicator, but is provided for possible future use.

guardtime - Time in minutes which a REPL_INTEGRITY_TREE level **integrity** directory must be stable before a client is allowed to update from it. (See explanation on **integrity** in section 4.20.2). The default guard time is 2 minutes.

random - A value in seconds sent by a master to its clients so that the clients can use the 0 - **random** range to generate a random time to wait on before updating from the master.

The configuration parameters (except the **importpath** and **exportpath**) values are all dynamically settable.

These APIs can be called whether the Replicator service is running or not. If the Replicator service is already running, any modification (except to the **importpath** and **exportpath**) to the replication configuration takes effect immediately, and is persistent after the Replicator service has been stopped. If the Replicator service is not started, the parameters are stored as persistent information and will take effect when the Replicator service starts up. The service must be stopped to set the **importpath** and **exportpath** values.

The Replicator configuration APIs information structures are:

```
typedef struct _REPL_INFO_0 {
        DWORD           rp0_role;
        LPWSTR          rp0_exportpath;
        LPWSTR          rp0_exportlist;
        LPWSTR          rp0_importpath;
        LPWSTR          rp0_importlist;
        LPWSTR          rp0_logonusername;
```

```
        DWORD           rp0_interval;
        DWORD           rp0_pulse;
        DWORD           rp0_guardtime;
        DWORD           rp0_random;
} REPL_INFO_0, *PREPL_INFO_0, *LPREPL_INFO_0;

typedef struct _REPL_INFO_1000 {
        DWORD           rp1000_interval;
} REPL_INFO_1000, *PREPL_INFO_1000, *LPREPL_INFO_1000;

typedef struct _REPL_INFO_1001 {
        DWORD           rp1001_pulse;
} REPL_INFO_1001, *PREPL_INFO_1001, *LPREPL_INFO_1001;

typedef struct _REPL_INFO_1002 {
        DWORD           rp1002_guardtime;
} REPL_INFO_1002, *PREPL_INFO_1002, *LPREPL_INFO_1002;

typedef struct _REPL_INFO_1003 {
        DWORD           rp1003_random;
} REPL_INFO_1003, *PREPL_INFO_1003, *LPREPL_INFO_1003;
```

4.20.1.1. NetReplGetInfo

NetReplGetInfo retrieves the Replicator service configuration information.

Security

No special group membership is required to successfully execute **NetReplGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetReplGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information required. Only level 0 is valid.

bufptr ___On return a pointer to the return information structure is written in the address pointed to by bufptr.

4.20.2. NetReplSetInfo

NetReplSetInfo modifies the Replicator service configuration information.

Security

Admin or server operator group membership is required to successfully execute **NetReplSetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetReplSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information to set 0, 1000, 1001, 1002, and 1003 are valid levels.

buf ___A pointer to a buffer containing the configuration information structure which corresponds to the specified level.

parm_err ___Optional pointer to a DWORD to return the identifier of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.20.3. Replicator Export Directory APIs

The ReplExportDir APIs control top-level directories under the export path on the master. A user can create a new directory under the export path and the Replicator service will automatically replicate that directory. Or, a directory under the export path can be registered using **NetReplExportDirAdd**. When adding a directory to be replicated via these APIs, the replication controls (**integrity** and **extent**) are specified via the NetReplExportDirAdd API. If the directory is created in the file system and no replicator APIs are called, then the directory is treated as having file integrity and tree extent.

Integrity determines when a master updates a client. When **integrity** is REPL_INTEGRITY_FILE, the client gets a replica of a file within the directory when it is not in use (being changed or replicated). When **integrity** is set to REPL_INTEGRITY_TREE, every file and directory within the replicated directory must be stable for the amount of time specified by the **guardtime** parameter before the client is updated. **Extent** determines whether the entire tree within the directory is replicated (REPL_EXTENT_TREE) or only the files in the first-level directory is replicated (REPL_EXTENT_FILE).

The replication controls of each replicated directory can be examined using **NetReplExportDirGetInfo**, and dynamically modified using **NetReplExportDirSetInfo**. These control fields used to be specified in the REPL.INI file within each replicated directory on LAN Manager 2.x, and they were not dynamically settable. On NT, the REPL.INI file is not used and will be ignored in the replication process.

NetReplExportDirEnum returns a list of directories that are currently replicated. **NetReplExportDirDel** deregisters a directory so that it is no longer replicated.

The lock status information is returned in two fields: **lockcount** and **locktime**. **Lockcount** indicates the number of outstanding locks on a directory. **Locktime** is the time (in seconds since 1970, GMT) when the directory was first locked, or is 0 if the directory is not locked at the present time.

NetReplExportDirLock locks a directory so that it is not replicated, by incrementing a lock reference count for the directory. A lock on a directory can be unlocked using **NetReplExportDirUnlock**. The replication does not resume unless all outstanding locks on that directory are released, and the lock reference count is returned to 0. (The **locktime** field is automatically set to 0 when **lockcount** is 0.)

The ReplExportDir APIs can be called whether the Replicator service is running or not. If the Replicator service is running as a master, any modification to the directory controls takes effect immediately, and is persistent after the Replicator service has been stopped. If the Replicator service is not started, the controls for the directory is stored as persistent information and will take effect when the Replicator service starts up.

The ReplExportDir APIs are available at the following information levels:

```
typedef struct _REPL_EDIR_INFO_0 {
        LPWSTR          rped0_dirname;
} REPL_EDIR_INFO_0, *PREPL_EDIR_INFO_0, *LPREPL_EDIR_INFO_0;

typedef struct _REPL_EDIR_INFO_1 {
        LPWSTR          rped1_dirname;
        DWORD           rped1_integrity;
        DWORD           rped1_extent;
} REPL_EDIR_INFO_1, *PREPL_EDIR_INFO_1, *LPREPL_EDIR_INFO_1;

typedef struct _REPL_EDIR_INFO_2 {
        LPWSTR          rped2_dirname;
        DWORD           rped2_integrity;
        DWORD           rped2_extent;
        DWORD           rped2_lockcount;
        DWORD           rped2_locktime;
} REPL_EDIR_INFO_2, *PREPL_EDIR_INFO_2, *LPREPL_EDIR_INFO_2;

typedef struct _REPL_EDIR_INFO_1000 {
        DWORD           rped1000_integrity;
} REPL_EDIR_INFO_1000, *PREPL_EDIR_INFO_1000,
*LPREPL_EDIR_INFO_1000;

typedef struct _REPL_EDIR_INFO_1001 {
        DWORD           rped1001_extent;
} REPL_EDIR_INFO_1001, *PREPL_EDIR_INFO_1001,
*LPREPL_EDIR_INFO_1001;
```

4.20.3.1. NetReplExportDirAdd

NetReplExportDirAdd registers an existing directory in the export path to be replicated. The default values for **locktime** and **lockcount** (both 0) are assumed.

Security

Admin or server operator group membership is required to successfully execute **NetReplExportDirAdd**.

NET_API_STATUS NET_API_FUNCTION
NetReplExportDirAdd (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information to add. Only level 1 is valid.

buf ___A pointer to a buffer containing the directory control information structure.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.20.4. NetReplExportDirDel

NetReplExportDirDel deregisters a replicated directory.

Privilege Level

Security

Admin or server operator group membership is required to successfully execute **NetReplExportDirDel**.

NET_API_STATUS NET_API_FUNCTION
NetReplExportDirDel (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *dirname*
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname ___A pointer to a NULL terminated Unicode string containing the name of a replicated directory to deregister.

4.20.5. NetReplExportDirEnum

NetReplExportDirEnum lists the replicated directories in the export path.

Security

No special group membership is required to successfully execute **NetReplExportDirEnum**.

NET_API_STATUS NET_API_FUNCTION
NetReplExportDirEnum (
IN LPWSTR *servername* **OPTIONAL**,
IN DWORD *level*,
OUT LPBYTE * *bufptr*,
IN DWORD *prefmaxlen*,
OUT LPDWORD *entriesread*,
OUT LPDWORD *totalentries*,
IN OUT LPDWORD *resumehandle* **OPTIONAL**
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information to return. Levels 0,1, and 2 are valid.

bufptr ___On return, an array of the specified information structure is returned in the address pointed to by bufptr.

prefmaxlen ___Preferred maximum length of returned data (in 8-bit bytes). A value of 0xFFFFFFFF indicates that all available entries should be returned.

entriesread ___On return, the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries ___On return, the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle ___On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing use search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored.

4.20.6. NetReplExportDirGetInfo

NetReplExportDirGetInfo retrieves a replicated directory control information.

Security

No special group membership is required to successfully execute **NetReplExportDirGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetReplExportDirGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *dirname*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname ___A pointer to a NULL terminated Unicode string containing the directory name to return control information about.

level ___Level of information to return. Levels 0, 1, and 2 are valid.

bufptr ___On return a pointer to the return information structure is written in the address pointed to by bufptr.

4.20.7. NetReplExportDirSetInfo

NetReplExportDirSetInfo modifies the control information of a replicated directory.

Security

Admin or server operator group membership is required to successfully execute **NetReplExportDirSetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetReplExportDirSetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *dirname*,
IN DWORD *level*,
IN LPBYTE *buf*,
OUT LPDWORD *parm_err* **OPTIONAL**
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname ___A pointer to a NULL terminated Unicode string containing the directory name to return control information about.

level ___Level of information to set. Levels 1, 1000, and 1001 are valid.

buf ___A pointer to a buffer containing the control information structure which corresponds to the specified level.

parm_err ___Optional pointer to a DWORD to return the identifier of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.20.7.1. NetReplExportDirLock

NetReplExportDirLock locks a replicated directory so that replication from it can be suspended. This function increments the lock reference count for the specified directory.

Security

Admin or server operator group membership is required to successfully execute **NetReplExportDirLock**.

NET_API_STATUS NET_API_FUNCTION
NetReplExportDirLock (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *dirname*
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname ___A pointer to a NULL terminated Unicode string containing the directory name to lock.

4.20.7.2. NetReplExportDirUnlock

NetReplExportDirUnlock unlocks a directory so that replication from it can resume. This function decrements the lock reference count for the specified directory.

Security

Admin or server operator group membership is required to successfully execute **NetReplExportDirUnlock**.

NET_API_STATUS NET_API_FUNCTION
NetReplExportDirUnlock (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *dirname*,
IN DWORD *unlockforce*
);

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname ___A pointer to a NULL terminated Unicode string containing the directory name to unlock.

unlockforce ___A value which indicates the force level to unlock the directory.

Force levels:

REPL_UNLOCK_NOFORCE - Unlocks the directory by decrementing the lock reference count. The lock reference count may or may not return to 0, so the directory could still be locked.

REPL_UNLOCK_FORCE - Unlocks the directory completely by removing all outstanding locks on the directory. The lock reference count is set to 0.

4.20.8. Replicator Import Directory APIs

The ReplImportDir APIs designate the top-level directories under the import path to receive updates on. They also return status information about a replicated directory on the client. On LAN Manager 2.x, a user must create a directory under the import path and the Replicator service automatically replicates to it. On NT, import directories are automatically added if they are exported by an export server from which importer is importing. Another way to register a directory in advance of it being exported, is to use the **NetReplImportDirAdd** API. This API does not create the directory itself. This is useful if you wish to modify some of the properties of the import directory (for example, to lock it) prior to it first beginning to import this directory.

NetReplImportDirDel deregisters a directory. This is used to clean up a directory that is no longer being exported. It will not stop replication if there is an active exporter, since it will be re-registered the next time the exporter tells the importer what directories it is exporting. If you wish to prevent importing of an actively exported directory, use the **NetReplImportDirLock** API.

NetReplImportDirEnum lists all the directories that are replicated to a client, and **NetReplImportDirGetInfo** returns the status of a specified directory.

The status information of a directory consists of the replication **state**, the UNC computername of the master (**mastername**), and the time (in seconds since 1970, GMT) when the directory was last updated (**last_update_time**). If the **state** is REPL_STATE_OK, the directory currently has a master, and is receiving regular update notices from it. If the **state** is REPL_STATE_NO_MASTER, the directory is not supported by any master, and it is normally empty. If the **state** is REPL_STATE_NO_SYNC, the directory has a master, but the master has not sent any update notices within the **interval** time period. This may be due to a communication failure, the master crashing, the directory being locked, files in the client directory being opened at update time, or an unstable REPL_INTEGRITY_TREE **integrity** directory on the master. If the client Replicator service is not started the **state** is REPL_STATE_NEVER_REPLICATED, **mastername** is a NULL string, and **last_update_time** is 0.**NetReplImportDirLock** locks a directory so that it does not receive updates, by incrementing a lock reference count for the directory. A lock on a directory can be unlocked using **NetReplImportDirUnlock**. The directory is not updated unless all outstanding locks on that directory are released, and the lock reference count is returned to 0.

The lock status information is returned in two fields: **lockcount** and **locktime**. **Lockcount** indicates the number of outstanding locks on a directory. **Locktime** is the time (in seconds since 1970, GMT) when the directory was first locked. (**locktime** is set to 0 whenever **lockcount** goes to 0.)

The ReplImportDir APIs can be called whether the Replicator service is running or not. If the Replicator service is running as a client, directory adds or deletes take effect immediately, and is persistent after the Replicator service has been stopped. If the Replicator service is not started, any added directory will receive updates when the Replicator service starts up (if there exists a master which exports the directory).

The ReplImportDir APIs are available at the following information levels:

```
typedef struct _REPL_IDIR_INFO_0 {
        LPWSTR          rpid0_dirname;
} REPL_IDIR_INFO_0, *PREPL_IDIR_INFO_0, *LPREPL_IDIR_INFO_0;

typedef struct _REPL_IDIR_INFO_1 {
        LPWSTR          rpid1_dirname;
        DWORD           rpid1_state;
        LPWSTR          rpid1_mastername;
        DWORD           rpid1_last_update_time;
        DWORD           rpid1_lockcount;
        DWORD           rpid1_locktime;
} REPL_IDIR_INFO_1, *PREPL_IDIR_INFO_1, *LPREPL_IDIR_INFO_1;
```

4.20.8.1. NetReplImportDirAdd

NetReplImportDirAdd registers an existing directory in the import path to receive replication from a master.

Security

Admin or replicator group membership is required to successfully execute **NetReplImportDirAdd**.

```
NET_API_STATUS NET_API_FUNCTION
NetReplImportDirAdd (
        IN LPWSTR servername OPTIONAL,
        IN DWORD level,
        IN LPBYTE buf,
        OUT LPDWORD parm_err OPTIONAL
        );
```

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level ___Level of information to add. Only level 0 is valid.

buf ___A pointer to a buffer containing the directory name.

parm_err ___Optional pointer to a DWORD to return the index of the first parameter in error when ERROR_INVALID_PARAMETER is returned. If NULL the parameter is not returned on error.

4.20.9. NetReplImportDirDel

NetReplImportDirDel deregisters directory so that it no longer receives updates from the master. Note that this API does not actually delete the directory from the file system. Also, the directory may be automatically re-registered by the Replicator service at any time. To prevent importing of a directory that is being exported by some Replicator service, use the **NetReplImportDirLock** API instead.

Privilege Level

Security

Admin or replicator group membership is required to successfully execute **NetReplImportDirDel**.

```
NET_API_STATUS NET_API_FUNCTION
NetReplImportDirDel (
    IN LPWSTR servername OPTIONAL,
    IN LPWSTR dirname
    );
```

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname ___A pointer to a NULL terminated Unicode string containing the name of a replicated directory to deregister.

4.20.10. NetReplImportDirEnum

NetReplImportDirEnum lists the replicated directories in the import path.

Security

No special group membership is required to successfully execute **NetReplImportDirEnum**.

```
NET_API_STATUS NET_API_FUNCTION
NetReplImportDirEnum (
    IN LPWSTR servername OPTIONAL,
    IN DWORD level,
    OUT LPBYTE * bufptr,
    IN DWORD prefmaxlen,
    OUT LPDWORD entriesread,
    OUT LPDWORD totalentries,
    IN OUT LPDWORD resumehandle OPTIONAL
    );
```

Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

level __Level of information to return. Levels 0 or 1 are valid.

bufptr __On return, an array of the specified information structure is returned in the address pointed to by bufptr.

prefmaxlen __Preferred maximum length of returned data (in 8-bit bytes). A value of 0xFFFFFFFF indicates that all available entries should be returned.

entriesread __On return, the actual enumerated element count is located in the DWORD pointed to by entriesread.

totalentries __On return, the total number of entries that could have been enumerated from the current resume position is located in the DWORD pointed to by totalentries.

resumehandle __On return, a resume handle is stored in the DWORD pointed to by resumehandle, and is used to continue an existing use search. The handle should be zero on the first call and left unchanged for subsequent calls. If resumehandle is NULL, then no resume handle is stored.

4.20.11. NetReplImportDirGetInfo

NetReplImportDirGetInfo retrieves the status information on a client replicated directory.

Security

No special group membership is required to successfully execute **NetReplImportDirGetInfo**.

NET_API_STATUS NET_API_FUNCTION
NetReplImportDirGetInfo (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *dirname*,
IN DWORD *level*,
OUT LPBYTE * *bufptr*
);

Parameters:

servername __A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname __A pointer to a NULL terminated Unicode string containing the directory name to return control information about.

level __Level of information to return. Levels 0 and 1 are valid.

bufptr __On return a pointer to the return information structure is written in the address pointed to by bufptr.

4.20.11.1. NetReplImportDirLock

NetReplImportDirLock locks a replicated directory so that replication to it can be suspended. This function increments the lock reference count for the specified directory.

Security

Admin or replicator group membership is required to successfully execute **NetReplImportDirLock**.

NET_API_STATUS NET_API_FUNCTION
NetReplImportDirLock (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *dirname*
);
Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname ___A pointer to a NULL terminated Unicode string containing the directory name to lock.

4.20.11.2. NetReplImportDirUnlock

NetReplImportDirUnlock unlocks a directory so that replication to it can resume. This function decrements the lock reference count for the specified directory.

Security

Admin or replicator group membership is required to successfully execute **NetReplImportDirUnlock**.

NET_API_STATUS NET_API_FUNCTION
NetReplImportDirUnlock (
IN LPWSTR *servername* **OPTIONAL**,
IN LPWSTR *dirname*,
IN DWORD *unlockforce*
);
Parameters:

servername ___A pointer to a NULL terminated Unicode string containing the name of the remote server on which the function is to execute. A NULL pointer or string specifies the local machine.

dirname ___A pointer to a NULL terminated Unicode string containing the directory name to unlock.

unlockforce ___A value which indicates the force level to unlock the directory.

Force levels:

REPL_UNLOCK_NOFORCE - Unlocks the directory by decrementing the lock reference count. The lock reference count may or may not return to 0, so the directory could still be locked.

REPL_UNLOCK_FORCE - Unlocks the directory completely by removing all outstanding locks on the directory. The lock reference count is set to 0.

5. Lanman APIs That Are Not Supported In Windows NT

The following section lists the 16-bit LANMan APIs that are no longer supported in the 32-bit API set. In general, the rule we have followed is to not support anything that was marked as obsolete in LANMan 2.x. In addition, there are some APIs that no longer make sense in NT, and those have been removed as well. Another set of APIs has been folded into the NT base or Win32 API set .

5.1. APIs with no 32 bit Net equivalents

- NetAlertStop, NetAlertStart
- NetBiosClose, NetBiosEnum, NetBiosGetInfo, NetBiosOpen, NetBiosSubmit
- NetCharDev and NetCharDevQ APIs
- NetErrorLogWrite
- NetLogonEnum
- NetHandleGetInfo levels 2 and 3
- NetMessageFileSend, NetMessageLogFileGet, NetMessageLogFileSet, NetMessageNameFwd, NetMessageNameUnFwd
- NetRemoteCopy, NetRemoteExec, NetRemoteMove
- NetServerAdminCommand
- NetWkstaSetUID

5.2. APIs that only have support for remoting to downlevel

- NetAccess APIs
- NetAudit APIs
- NetConfig APIs
- NetError APIs

6. Interoperabilty Considerations

6.1. Requests From 16-bit LANMan Clients

NT provides support for most remote API called from downlevel clients. However, the following calls are **NOT** supported when remoted from a downlevel client to an NT server. In some of the below, there is a more current "2" version of the API which is supported (e.g. NetServerEnum).

NetFileEnum
NetFileGetInfo
NetFileClose
NetServerAdminCommand
NetAuditOpen
NetAuditClear
NetErrorLogOpen
NetErrorLogClear
NetMessageNameFwd
NetMessageNameUnFwd
NetMessageFileSend
NetMessageLogFileSet
NetMessageLogFileGet
NetUserAdd
NetUserSetInfo
NetUserPasswordSet
NetWkstaSetUID
NetUseEnum
NetUseAdd
NetUseDel
NetUseGetInfo
NetProfileSave
NetProfileLoad
NetStatisticsGet
NetStatisticsClear
NetNetBiosEnum
NetNetBiosGetInfo
NetServerEnum
NetConfigGet2
NetConfigGetAll2
NetHandleGetInfo
NetHandleSetInfo
NetAuditRead
NetUserValidate2
NetAccessCheck
NetAlertRaise
NetAlertStart
NetAlertStop
NetAuditWrite
NetServiceStatus
DosPrintDriverEnum
DosPrintQProcessorEnum
DosPrintPortEnum
DosPrintDest
NetConfigSet

6.2. Calling 16-bit LANMan Servers

When an RPC based API fails to connect to the appropriate interface the client-side stub may attempt to initiate a down-level API request to the server selected. For most of the Windows networking APIs specified in this document, and any API where the functionality and data formats are changed only for 32-bit usage, the conversion is straightforward. For components which offer new functionality the caller of the API should generally be aware of the destination type. When the new API offers a superset of

the functionality of the down-level station the same API is used for both destinations, but the new API fields must have either a reserved value of an associated field to inform the conversion layer the field may be ignored if going downlevel. This is required so that an API caller is not misled as to the action performed when the API was called.