Appendix 4A ConnMan API

Contents

CONNAuthenticate

Description

Authenticates a *connHandle* without using an *authHandle*.

Syntax

```
UINTXX DIST
CONNAuthenticate (
    UINT32              processGroupID,
    UINT32              processId,
    CONN_HANDLE  connHandle,
    UINT32              authFlags,
    UINT32 DIST         *authSvcId,
    SPECT_DATA DIST  *userName,
    SPECT_DATA DIST  *password,
    SPECT_DATA DIST  *domainName,
    VOID DIST           *pAuthSpecInfo)
```

Input

processGroupID	Calling function's group ID.
processID	Calling function's process ID.
connHandle	The connection to authenticate.
authFlags	Determines whether the password should be prompted for from a secure ring-0 environment. Possible values for this field are: CONN_PASSWD_PROMPT_NONE CONN_PASSWD_PROMPT
authSvcId	The unique ID of the authentication service to use in creating this authentication handle. Must be one of these values: AUTH_SVC_BINDERY_ID AUTH_SVC_NDS_ID AUTH_SVC_PNW_ID
userName	Pointer to the user name to use in authenticating the connection. The SPECT_DATA fields must be correctly filled in (the *Data* buffer must contain the user name and the length field must be correct).

password	A collection of bytes representing the password. It is specified in a SPECT_DATA structure by filling out the length field of the string type and pointing the *Data* field at the password buffer.
domainName	Pointer to the domain name where the authentication credentials are valid so they can be used in authenticating the connection. The SPECT_DATA fields must be correctly filled out (that is, the *Data* buffer must contain the domain name and the *Length* field must be correct).
pAuthSpecInfo	Pointer to any specific information required by the authentication service. The first DWORD of this pointer should contain the number of bytes of this buffer that contain information.

Output

None.

Remarks

This function authenticates a connection without first creating an authentication handle. It therefore requires that all of the information that is needed to authenticate a connection be explicitly passed in.

This function determines if the *connHandle* has previously been authenticated. If it has, the function returns an error. If it hasn't been authenticated, the function will call down to the authentication multiplexor to authenticate the connection using the given authentication information.

This function will not pass back the authentication handle that has been created.

See also

CONNAuthenticateWithHandle
CONNUnauthenticate
CONNCreateAuthenticationHandle
CONNDestroyAuthenticationHandle
CONNScanAuthenticationHandles
CONNGetAuthHandleInfo

CONNAuthenticateWithHandle

Description	Authenticates a *connHandle* using an *authHandle*.

Syntax

```
UINTXX DIST
CONNAuthenticateWithHandle (
    AUTH_HANDLE   authHandle,
    CONN_HANDLE  connHandle)
```

Input

authHandle	The authentication handle to use when authenticating this connection.
connHandle	The connection to authenticate.

Output

None.

Remarks

This function determines if the *connHandle* has previously been authenticated. If it has, it will return an error. If it hasn't, it will call down to the authentication multiplexor to authenticate the connection with the specified authentication handle.

See also

CONNAuthenticate
CONNUnauthenticate
CONNCreateAuthenticationHandle
CONNDestroyAuthenticationHandle
CONNScanAuthenticationHandles
CONNGetAuthHandleInfo
CONNChangePassword
CONNVerifyPassword

CONNChangePassword

Description

Synchronizes a password change across a domain (consisting of several bindery servers, and/or several trees). The caller specifies whether this function uses a dialog box requesting the old and new passwords (allowing for greater security to be built into applications).

Syntax

```
UINTXX DIST
CONNChangePassword (
    UINT32                  authHandle,
    UINT32                  authFlags,
    SPECT_DATA DIST *oldPassword,
    SPECT_DATA DIST *newPassword)
```

Input

authHandle	Authentication handle to set the password for.
flags	Controls whether a secure prompting for the password is made from ring-0. The flags may have one of the following values: CONN_PASSWD_PROMPT_NONE CONN_PASSWD_PROMPT_NEW CONN_PASSWD_PROMPT_OLD CONN_PASSWD_PROMPT_BOTH
oldPassword	Old password, stored in SPECT_DATA structure. It must be correctly initialized. If the password is to be prompted for from ring 0, this parameter should be set to NULL.
newPassword	New password, stored in SPECT_DATA structure. It must be correctly initialized. If the password is to be prompted for from ring 0, this parameter should be set to NULL.

Output

None.

See also	CONNAuthenticateWithHandle CONNAuthenticate CONNUnauthenticate CONNCreateAuthenticationHandle CONNDestroyAuthenticationHandle CONNScanAuthenticationHandles CONNGetAuthHandleInfo CONNVerifyPassword

CONNClose

Description

Closes the connection with the specified *connHandle*. This call is made when the caller that has previously opened the connection has finished using it.

Syntax

```
UINTXX DIST
CONNClose (
    UINT32              processGroupID,
    UINT32              processId,
    CONN_HANDLE connHandle,
    UINT32              flags)
```

Input

processGroupID	Calling function's process group ID.
processID	Calling function's process ID.
connHandle	The connection handle to be closed.
flags	LONG_LIVED_CONNECTION. This connection was opened as a long-lived connection, and should now be terminated even if other applications are using it. SHORT_LIVED_CONNECTION. The connection was opened as a short-lived connection, and should be terminated only if no other applications are using it.

Output

None.

Remarks

After all open handles to a connection are closed, the connection is either destroyed or else placed upon a list of disposable connections for later reference. If a connection is to be destroyed, the appropriate **SESSDisconnect** routine is called to destroy the connection.

If other processes are still using this connection, simply decrement the *in-use* count and leave the connection alone.

Any connection that is placed on the disposable list may be either reopened in the future (if a connection open request matching the disposed connection is received), or else destroyed (if an algorithm determines that reusing old disposable connections is a more efficient use of memory than allocating new memory for a new connection).

If other processes are still have this connection open, the *in-use* count is simply decremented to reflect that this process has closed the connection.

See also

CONNOpenByAddress
CONNOpenByName
CONNOpenPreferred
CONNOpenByReference

CONNCreateAuthenticationHandle

Description

Creates an authentication handle.

Syntax

```
UINTXX DIST
CONNCreateAuthenticationHandle (
    UINT32                  processGroupID,
    UINT32                  processId,
    UINT32                  authFlags,
    UINT32 DIST            *authSvcId,
    SPECT_DATA DIST        *userName,
    SPECT_DATA DIST        *password,
    SPECT_DATA DIST        *domainName,
    VOID DIST              *pAuthSpecInfo,
    AUTH_HANDLE DIST *authHandle)
```

Input

processGroupID	Calling function's process group ID
processID	The process identifier to associate with the connection.
authFlags	Determines whether to prompt for a password from a secure ring-0 environment. Possible values for this field include the following: CONN_PASSWD_PROMPT_NONE CONN_PASSWD_PROMPT
authSvcId	The unique ID of the authentication service to use to create this authentication handle. It must be one of the following values: AUTH_SVC_BINDERY_ID AUTH_SVC_NDS_ID AUTH_SVC_PNW_ID
userName	Pointer to the username to use in creating the authentication handle. The SPECT_DATA fields must be correctly filled out (that is, the *Data* buffer must contain the user name and the *Length* field must be correct).

password	A collection of bytes representing the password. It is specified in a SPECT_DATA structure by filling out the *Length* field of the string type and pointing the *Data* field at the password buffer.
domainName	Pointer to the domain name where the authentication credentials are valid. The credentials are used to authenticate the connection. The SPECT_DATA fields must be correctly filled out (that is, the *Data* buffer must contain the domain name and the *Length* field must be correct).
pAuthSpecInfo	Pointer to any specific information required by the authentication service. The first DWORD of this pointer should contain the number of bytes of this buffer containing information.

Output

authHandle	The created authentication handle is returned here.

Remarks

All necessary information is supplied as parameters to the call. The authentication service is called (through the AuthMux) to perform the actual creation of the authentication handle.

See also

CONNAuthenticateWithHandle
CONNAuthenticate
CONNUnauthenticate
CONNDestroyAuthenticationHandle
CONNScanAuthenticationHandles
CONNGetAuthHandleInfo
CONNChangePassword
CONNVerifyPassword

CONNDecInfo

Description

Decrements a *connHandle* counter.

Syntax

```
UINTXX DIST
CONNDecInfo (
    CONN_HANDLE connHandle,
    UINT32            infoId)
```

Input

connHandle	The connection handle of the desired connection.
infoId	Specifies the connection information which should be changed. It can be the following: CONN_ENTRY_RESOURCE_COUNT

Output

None.

Remarks

This function is reserved for system NLMs that are tracking resources. It allows them to decrement the connection's resource count to indicate that the connection is no longer in use.

See also

CONNIncInfo

CONNDestroyAuthenticationHandle

Description

Destroys an authentication handle.

Syntax

```
UINTXX DIST
CONNDestroyAuthenticationHandle (
    AUTH_HANDLE authHandle)
```

Input

authHandle	The authentication handle to destroy.

Output

None.

Remarks

This function finds all connection handles that use the specified authentication handle and then calls down to the authentication multiplexor to unauthenticate those connections. After they have all been unauthenticated, a call to the authentication multiplexor will destroy the authentication handle.

See also

CONNAuthenticateWithHandle
CONNAuthenticate
CONNUnauthenticate
CONNCreateAuthenticationHandle
CONNScanAuthenticationHandles
CONNGetAuthHandleInfo
CONNChangePassword
CONNVerifyPassword

CONNGetAuthHandleInfo

Description	Returns information on a given authentication handle.

Syntax

```
UINTXX DIST
CONNGetAuthHandleInfo (
    AUTH_HANDLE    authHandle,
    UINT32 DIST          *authSvcId,
    SPECT_DATA DIST  *userName,
    SPECT_DATA DIST  *domainName,
    VOID DIST              *pAuthSpecInfo)
```

Input

authHandle	Authentication handle for which to return information.

Output

authSvcId	Unique ID of the authentication service used to create this authentication handle. It must be one of the following values: AUTH_SVC_BINDERY_ID AUTH_SVC_NDS_ID AUTH_SVC_PNW_ID
userName	Pointer to the buffer containing the user name used in creating this authentication handle. The SPECT_DATA fields must be correctly filled out (that is, the *Data* buffer must have sufficient size to receive the username and the *Length* field must be filled in when this function is called).
domainName	Pointer to the buffer containing the domain name used in creating this authentication handle. The SPECT_DATA fields must be correctly filled out (that is, the *Data* buffer must have sufficient size to receive the domainName and the *Length* field must be filled in when this function is called).

pAuthSpecInfo
: Pointer to any specific information set by the authentication service. The first DWORD of this pointer should contain the number of bytes of buffer space available to store returned information.

Remarks

This call returns the same information about an authentication handle as **CONNScanAuthenticationHandles**, but can be used to identify information specific to a given authentication handle without scanning until that authentication handle is identified.

See also

CONNAuthenticateWithHandle
CONNAuthenticate
CONNUnauthenticate
CONNCreateAuthenticationHandle
CONNDestroyAuthenticationHandle
CONNScanAuthenticationHandles
CONNChangePassword
CONNVerifyPassword

CONNGetDefaultConnection

Description

Return the default connection handle associated with a process and process group.

Syntax

```
UINTXX DIST
CONNGetDefaultConnection (
    UINT32      processGroupID,
    UINT32      processId,
    CONN_HANDLE DIST *connHandle)
```

Input

processGroupID	Calling function's process group ID.
processID	Process identifiers to associate with the connection.

Output

connHandle	The connection handle to associate with the specified process identifiers.

See also

CONNSetDefaultConnection

CONNGetNumConnections

Description

Returns the number of currently allocated connection entries. The value returned reflects the total number of connections possible, including those currently in use.

Syntax

```
UINTXX DIST
CONNGetNumConnections (
    UINT32 DIST *numberOfEntries)
```

Input

None.

Output

numberOfEntries	The number of connection entries that have been allocated.

Remarks

ConnMan will return the number of connection entries which are currently allocated. Some of these connections may be private and thus would not be visible to all processes.

Because the connection table is dynamically extensible at run-time, the call should not hold on to this value The number of connection entries which have been allocated is a dynamic value and will change over time; the caller should not assume that the value returned will remain the same.

See also

None.

CONNGetStructure

Description

Returns structure-type connection information for a given connection handle. The caller must allocate enough space to receive a copy of the information.

Syntax

```
UINTXX DIST
CONNGetStructure (
    CONN_HANDLE  connHandle,
    UINT32        infoId,
    UINT32        infoLen,
    VOID DIST     *infoPtr)
```

Input

connHandle
Connection handle

infoId The connection parameter, which can be one of the following:

Value	Data type	Meaning
CONN_ENTRY_TRAN_ADDR	TRAN_ADDR_TYPE	Transport address
CONN_ENTRY_DOMAIN_NAME	SPECT_DATA	Connection domain name
CONN_ENTRY_SERVER_NAME	SPECT_DATA	Connection server name
CONN_ENTRY_SERVICE_NAME	SPECT_DATA	Connection service name
CONN_ENTRY_RETURN_ALL	CONN_INFO_TYPE	Return the whole structure

All of these items may be queried by calls external to the client.

infoLen Length of output buffer into which to return information.

If the structure is a TRAN_ADDR_TYPE, the *infoLen* field should be the size of that structure.

If the structure is a SPECT_DATA, the *infoLen* field should be the size of a SPECT_DATA structure. In addition, the name field of the structure should already be filled in with a pointer to a buffer of size SPECT_DATA.*Length*.

This buffer will receive the name value of the SPECT_DATA field, which can be predetermined by calling **CONNQueryStringLength**. If this value is less than required to copy the *Data* field, an error will be returned after copying the portion which will fit into the *infoPtr* buffer.

For example, pretend that the caller wants to get the value of the server name for a connection.

Step 1. Determine the size of buffer needed to store the name by calling **CONNQueryStringLength**, thus:

```
CONNQueryStringLength (connHandle,
CONN_ENTRY_SERVER_NAME, &nameLength);
```

Step 2. Allocate space for the name.

```
serverName.Data = NIOSShortTermAlloc
(modHandle, nameLength);
serverName.Length = nameLength;
serverName.DataType = SPECT_DATA_ASCII;
serverName.CountryCode = 0;
serverName.LocalCodePage = 0;
```

Step 3. Get the name itself with **CONNGetStructure**.

```
CONNGetStructure(connHandle,
CONN_ENTRY_SERVER_NAME, sizeof
(SPECT_DATA_TYPE), &serverName);
```

If the *infoId* is CONN_ENTRY_RETURN_ALL, then the *infoLen* parameter should be the size of the CONN_INFO_TYPE. This structure size does not reflect the size of the variable string *Data* parameters of the SPECT_DATA entries. These pointers should be pre-initialized to buffers which are sized correctly to receive the variable length string.

CONNQueryStringLength can be used to pre-determine the correct size. If any of these SPECT_DATA buffers are too small, an error will be returned.

Output *infoPtr* Pointer to the buffer into which to receive information.

If the structure requested is a SPECT_DATA structure, it must have a valid pointer already in the *Data* field that has enough room to hold the name.

Remarks

The caller can get one piece of the connection information structure or the whole structure. Some of the entries in the structure are pointers. The caller must fill in the pointer to a valid data area that is large enough for the Requester to copy the information into. If the caller specifies CONN_ENTRY_RETURN_ALL and doesn't want all the SPECT_DATA information strings, a NULL can be passed in for the particular field that is not desired.

An error is returned if the output buffer is too small to receive the requested information.

See also

CONNGetValue
CONNSetStructure
CONNSetValue
CONNScanInfo

CONNGetValue

Description

Returns specific *value* (as opposed to structure) connection information for the given connection handle.

Syntax

```
UINTXX DIST
CONNGetValue (
    CONN_HANDLE connHandle,
    UINT32         infoId,
    VOID DIST      *infoPtr)
```

Input

connHandle	Connection handle.
infoId	Type of information to be returned can be one of the following:

*Avail	Value	Data type	Meaning
A	CONN_ENTRY_VERSION	UINT32	Version of CONN_INFO struct
A	CONN_ENTRY_AUTH_USER_ID	UINT32	Id of user authenticated as
A	CONN_ENTRY_AUTH_SVC_ID	UINT32	Id of authentication module:
			AUTH_SVC_BINDERY_ID
			AUTH_SVC_NDS_ID
			AUTH_SVC_PNW_ID
A	CONN_ENTRY_AUTH_HANDLE	UINT32	Authentication Handle
I	CONN_ENTRY_AUTH_SPEC_PTR	UINT32	Pointer to auth-specific info
A	CONN_ENTRY_SESS_SVC_ID	UINT32	Session Protocol Provider Id:
			NCP_SESSION_ID
			SMB_SESSION_ID
I	CONN_ENTRY_SESS_SPEC_PTR	UINT32	Pointer - Session-specific info
A	CONN_ENTRY_NAME_SVC_ID	UINT32	Id of name service provider:
			NAME_SVC_BINDERY_ID
			NAME_SVC_NDS_ID
			NAME_SVC_PNW_ID
A	CONN_ENTRY_MAX_IO	UINT32	Maximum IO for connection
A	CONN_ENTRY_MAX_RW_IO	UINT32	Maximum read/write IO
A	CONN_ENTRY_ROUND_TRIP	UINT32	Round trip time in milliseconds
A	CONN_ENTRY_SECURITY	UINT32	Security mode in effect
			Bit definitions:
			CFG_CRC
			CFG_MD4
			CFG_CRYPT
A	CONN_ENTRY_LICENSE	UINT32	License state of connection ??
I	CONN_ENTRY_TRAN_ADDR_OBJ	UINT32	Pointer to the tran addr object

I	CONN_ENTRY_NCP_HOOK_RTNS	UINT32	Pointer to NCP hook routines
A	CONN_ENTRY_SFT_LEVEL	UINT32	Current sft level
A	CONN_ENTRY_TTS_LEVEL	UINT32	Current tts level
A	CONN_ENTRY_SERVER_CONN_NUM	UINT32	Server connection number
A	CONN_ENTRY_SERVER_VERSION	UINT32	Server version
A	CONN_ENTRY_PERM	BIT	Permanent flag for connection
A	CONN_ENTRY_AUTH	BIT	Authenticated state
A	CONN_ENTRY_ANCHOR	BIT	Anchor state for connection
A	CONN_ENTRY_SUSPENDED	BIT	Suspended state for condition
A	CONN_ENTRY_TRAN_SVC_ID	UINT32	Transport Service Id
A	CONN_ENTRY_ORDER_NUM	UINT32	Connection order number
A	CONN_ENTRY_RETURN_ALL	CONN_ENTRY_INFO	
A	CONN_ENTRY_RETURN_NONE	n/a	

*A Available to all calling functions
I   Available to internal client NLMs only (that is, no external function should ever need to access these items).

Output *infoPtr* Pointer to the buffer which should receive the data. All bit fields are a UINT32 type. (Zero if clear, else set)

See also
CONNGetStructure
CONNSetStructure
CONNSetValue
CONNScanInfo

CONNIncInfo

Description

Increments a *connHandle* counter.

Syntax

```
UINTXX DIST
CONNIncInfo (
    CONN_HANDLE connHandle,
    UINT32 infoId)
```

Input

connHandle	The connection handle of the desired connection.
infoId	Connection information which should be changed. It can be the following: CONN_ENTRY_RESOURCE_COUNT

Output

None.

Remarks

This function is reserved for system NLMs that are tracking resources. It allows them to increment a connection's resource count to indicate that the connection is in use.

See also

CONNDecInfo

CONNOpenByAddress

Description

Calls the specified session protocol module to establish a connection with the remote entity specified by the transport address.

Syntax

```
UINTXX DIST
CONNOpenByAddress (
    UINT32     processGroupId,
    UINT32     processId,
    UINT32     flags,
    UINT32     sessionSvcId,
    TRAN_ADDR_TYPE DIST    *tranAddr,
    CONN_HANDLE DIST       *repConnHandle)
```

Input

processGroupID	Calling function's process group ID.
processID	Calling function's process ID.
flags	LONG_LIVED_CONNECTION. The connection should last past the termination of the calling process. SHORT_LIVED_CONNECTION. The connection should not remain past the termination of the calling process.
sessionSvcId	NCP_SESSION_ID SMB_SESSION_ID WILD_SESSION_ID Can be used alone or ORed with another *sessionSvcId*. If it is ORed with another ID, the other session service will be tried first. If that fails or if only a wild card is specified, the remaining session services will be tried according to their load order.
tranAddr	The destination transport address, correctly formatted for the transport type specified in this structure.

Output

repConnHandle	A pointer to the connection handle to be

returned. This connection handle may be used for all requests directed to this connection.

Remarks

If a connection already exists that matches the input *processGroupID*, *processId*, *sessionSvcId*, and *tranAddr*, then the *in-use* count of the already-established connection is incremented and a handle to that connection handle is returned.

ConnMan will either return the connection handle of an existing connection or else will call the **SESSConnectByAddress** routine of the corresponding session protocol module to establish a new connection to the remote entity. This will bind the connection both to a specific session protocol module and to a specific transport protocol module, thus allowing high-level API requests (such as **FileOpen**) to be multiplexed to the correct session protocol module (for example, NCP). Also, low-level API requests (such as **SendPacket**) used by session protocols will be multiplexed to the correct transport protocol module (such as IPX).

See also

CONNOpenByName
CONNOpenPreferred
CONNOpenByReference
CONNClose

CONNOpenByName

Description

Resolves a given name to a transport address/session protocol pair. The appropriate session protocol is then called to establish a connection using the transport address.

Syntax

```
UINTXX DIST
CONNOpenByName (
    UINT32               processGroupID,
    UINT32               processId,
    UINT32               flags,
    SPECT_DATA DIST      *name,
    UINT32               nameSvcId,
    SPECT_DATA DIST      *objectType,
    UINT32               tranSvcId,
    CONN_HANDLE DIST     *repConnHandle)
```

Input

processGroupID	Calling function's process group ID.
processID	Calling function's process ID.
flags	LONG_LIVED_CONNECTION. The connection should remain past the termination of the calling process
	SHORT_LIVED_CONNECTION. The connection should not remain past the termination of the calling process
name	Pointer to the user-readable name to resolve to a connection. The string must be NULL-terminated and a maximum of 512 characters. If this string is Unicode, then the string has a maximum of 1024 bytes, and the SPECT_DATA fields must be correctly filled out.

	nameSvcId	Desired name service ID. NAME_SVC_BINDERY_ID NAME_SVC_NDS_ID NAME_SVC_PNW_ID SVC_ID_WILDCARD Can be by itself or ORed with another *nameSvcId*. If it is ORed, the other name service will be tried first. If that name service fails or if only a wild card is specified, the remaining name services will be tried in the order specified in the NET.CFG protocol order.
	objectType	Address of desired object type. This will be one of the OBJECT_TYPE identifiers found in CLIENT32.H, but must be placed into a SPECT_DATA structure.
	tranSvcId	Desired transport ID. TRAN_ID_IPX TRAN_ID_UDP TRAN_ID_WILDCARD See explanation above.
Output	*repConnHandle*	A pointer to the connection handle to be returned. This connection handle may be used for all requests directed to this connection.

Remarks

If a connection already exists which matches the input *processGroupID*, *processID*, *name*, *nameSvcId*, *objectType*, and *tranSvcId*, the *in-use* count of the already-established connection is incremented and a handle to that connection is returned.

ConnMan will either return the connection handle of an existing connection with a matching name or else will call **NAMEResolveToAddress** to resolve the name to a transport address and session protocol.

ConnMan will use this address and session protocol to open a connection. Opening a connection will either return an existing connection handle or will call the corresponding session protocol module to establish a new connection to the remote entity. This will bind the connection both to a specific session protocol module and to a specific transport protocol module.

After the connection is established, high-level API requests (such as **FileOpen**) can be multiplexed to the correct session protocol module (such as NCP); low-level API requests (such as **SendPacket**) can be multiplexed to the correct transport protocol module (such as IPX).

See also

CONNOpenByAddress
CONNOpenByName
CONNOpenPreferred
CONNOpenByReference
CONNClose

CONNOpenByReference

Description

Opens a *connHandle* for a connection reference specified by the *connReference* parameter. (This reference was returned from a call to **CONNScanInfo**.)

Syntax

```
UINTXX DIST
CONNOpenByReference (
    UINT32                    processGroupID,
    UINT32                    processId,
    UINT32                    flags,
    UINT32                    connReference,
    CONN_HANDLE DIST *repConnHandle)
```

Input

processGroupID	Process group ID to associate with new connection.
processID	Process ID to associate with new connection.
flags	LONG_LIVED_CONNECTION. The connection should remain past the termination of the calling process. SHORT_LIVED_CONNECTION. The connection should not remain past the termination of the calling process.

Output

repConnHandle	A pointer to the connection handle to be returned. This connection handle may be used for all requests directed to this connection.

Remarks

connReference refers to an existing connection which was found by scanning connections for specific information. If the input parameters *processGroupID* and *processId* specify a private connection, then a new connection will be established to the remote entity; otherwise, the *in-use* count of the connection associated with the reference handle is incremented and a connection handle to that connection is returned.

Any connection that is returned will be bound to a specific session protocol module and to a specific transport protocol module, thus allowing high-level API requests (such as **FileOpen)** to be multiplexed to the correct session protocol module (for example, NCP), and low-level API requests (such as **SendPacket**) to be multiplexed to the correct transport protocol module (for example, IPX).

See also

CONNOpenByAddress
CONNOpenByName
CONNOpenPreferred
CONNOpenByReference
CONNClose
CONNScanInfo

CONNOpenPreferred

Description

Returns a *connHandle* to the preferred connection defined in the NET.CFG configuration file. The connection will be made to either the preferred server or to the preferred tree.

Syntax

```
UINTXX DIST
CONNOpenPreferred (
    UINT32                processGroupId,
    UINT32                processId,
    UINT32                flags,
    UINT32 DIST           *nameSvcId,
    UINT32                tranSvcId,
    CONN_HANDLE DIST  *repConnHandle)
```

Input

processGroupID	Process group ID to associate with new connection.
processID	Process ID to associate with new connection.
flags	LONG_LIVED_CONNECTION. The connection should remain past the termination of the calling process SHORT_LIVED_CONNECTION. The connection should not remain past the termination of the calling process
nameSvcId	NAME_SVC_BINDERY_ID NAME_SVC_NDS_ID NAME_SVC_PNW_ID SVC_ID_WILDCARD. Can be used by itself or ORed with another *nameSvcId*. If ORed with another ID, then the other name service will be tried first. If that fails or if only a wild card is specified, the remaining name services will be tried in the order specified in the NET.CFG protocol order.
tranSvcId	TRAN_ID_IPX TRAN_ID_UDP TRAN_ID_WILDCARD.

		Can be used by itself or ORed with another *tranSvcId*. If ORed with another ID, then the other transport service will be tried first. If that fails or if only a wild card is specified, the remaining transport services will be tried according to their load order.
Output	*repConnHandle*	A pointer to the connection handle being returned. This connection handle may be used for all subsequent requests directed to this connection.

Remarks

The algorithm used in this routine is as follows:

If a preferred name has been set:

1. Determine the preferred name by calling **NAMEGetPreferred**.
2. Resolve this name to an address using **NAMEResolveToAddress**.
3. Open a connection using the address, and receive back a connection handle.

If no preferred name has been set, or the preferred name cannot be resolved to an address:

1. Call **NAMEGetInitialConnection** to return any connection that can be found. Any connection that is returned will be bound to to a specific session protocol module and to a specific transport protocol module.

See also	CONNOpenByAddress CONNOpenByName CONNOpenPreferred CONNOpenByReference CONNClose

CONNQueryStringLength

Description

Returns the length of the variable portion of a SPECT_DATA item.

Syntax

```
UINTXX DIST
CONNQueryStringLength (
    CONN_HANDLE  connHandle,
    UINT32           infoId,
    UINT32 DIST      *stringLen )
```

Input

connHandle The connection handle.

infoId The SPECT_DATA item, which can be one of the following:

Value	Meaning
CONN_ENTRY_DOMAIN_NAME	Connection domain name
CONN_ENTRY_SERVER_NAME	Connection server name
CONN_ENTRY_SERVICE_NAME	Service type name

All of these items may be queried by calls external to the client.

Output

stringLen Length of output buffer required to store the variable portion of a SPECT_DATA object.

Remarks

This call will be made just prior to making a **CONNGetStructure** call, and will determine the correct size of buffer that will allow it to return all of the requested data.

See also

CONNGetStructure

CONNScanAuthenticationHandles

Description

Scans through authentication handles, determining which authentications exist within the caller's scope.

Syntax

```
UINTXX DIST
CONNScanAuthenticationHandles (
    UINT32                  processID,
    UINT32                  processId,
    UINT32 DIST             *scanHandle,
    AUTH_HANDLE DIST        *authHandle,
    UINT32 DIST             *authSvcId,
    SPECT_DATA DIST         *userName,
    SPECT_DATA DIST         *domainName,
    VOID DIST               *pAuthSpecInfo)
```

Input

processGroupID	Calling function's process group ID.
processID	Calling function's process ID.
scanHandle	Address of the handle to be used to retrieve the next authentication handle. This value should initially be set to zero. The output value of *scanHandle* will be the next handle to use on subsequent calls to this function.

Output

authSvcId	Unique ID of the authentication service used to create this authentication handle. It will be either AUTH_SVC_BINDERY_ID, AUTH_SVC_NDS_ID, or AUTH_SVC_PNW_ID.
userName	Pointer to the buffer in which to return the user name used in creating this authentication handle. The SPECT_DATA fields must be correctly filled out (that is, the *Data* buffer must have sufficient size to receive the username and the *Length* field must be filled in when this function is called).
domainName	Pointer to the buffer to return the domain name used in creating this authentication handle. The

SPECT_DATA fields must be correctly filled out (that is, the *Data* buffer must have sufficient size to receive the domainName and the *Length* field must be filled in when this function is called).

pAuthSpecInfo	Pointer to any specific information set by the authentication service. The first DWORD of this pointer should contain the number of bytes of buffer space available to store returned information.

See also

CONNAuthenticateWithHandle
CONNAuthenticate
CONNUnauthenticate
CONNCreateAuthenticationHandle
CONNDestroyAuthenticationHandle
CONNGetAuthHandleInfo
CONNChangePassword
CONNVerifyPassword

CONNScanInfo

Description

Returns connection information for multiple connections. It will return either one piece or the full structure of connection information for one connection at time.

Syntax

```
UINTXX DIST
CONNScanInfo (
    UINT32        processGroupID,
    UINT32        processId,
    UINT32 DIST  *scanReference,
    UINT32        scanInfoId,
    VOID DIST    *scanMatchPtr,
    UINT32        scanFlags,
    UINT32        retInfoId,
    UINT32        retInfoLen,
    VOID DIST    *retInfoPtr,
    UINT32 DIST *connReference)
```

Input

processGroupID	Calling function's process group ID.
processID	Calling function's process ID.
scanReference	The reference to be used on the next iteration of the scan. This value should be initially set to zero. The output of this parameter will be used in subsequent calls to this function.
scanInfoId	Specifies which connection information is to be scanned for. (Caller cannot specify matching the entire CONN_INFO_STRUCT).

The following table shows all available connection information.

Value	Data type	Meaning
CONN_ENTRY_AUTH_USER_ID	UINT32	Id of user authenticated
CONN_ENTRY_AUTH_SVC_ID	UINT32	Id of authentication module
CONN_ENTRY_AUTH_HANDLE	UINT32	Authentication Handle
CONN_ENTRY_AUTH_SPEC_PTR	UINT32	Pointer to auth specific info
CONN_ENTRY_SESS_SVC_ID	UINT32	Session Protocol Provider Id
CONN_ENTRY_SESS_SPEC_PTR	UINT32	Pointer - Session specific info
CONN_ENTRY_NAME_SVC_ID	UINT32	Id of name service provider
CONN_ENTRY_MAX_IO	UINT32	Maximum IO for connection
CONN_ENTRY_MAX_RW_IO	UINT32	Maximum read/write IO
CONN_ENTRY_ROUND_TRIP	UINT32	Round trip time in milliseconds
CONN_ENTRY_SECURITY	UINT32	Security mode in effect
CONN_ENTRY_LICENSE	UINT32	License state of connection
CONN_ENTRY_TRAN_ADDR_OBJ	UINT32	Pointer to the tran addr object
CONN_ENTRY_TRAN_SVC_ID	UINT32	Id of transport service provider
CONN_ENTRY_NCP_HOOK_RTNS	UINT32	Pointer to NCP hook routines
CONN_ENTRY_SFT_LEVEL	UINT32	Current sft level
CONN_ENTRY_TTS_LEVEL	UINT32	Current tts level
CONN_ENTRY_SERVER_CONN_NUM	UINT32	Server connection number
CONN_ENTRY_SERVER_VERSION	UINT32	Server version
CONN_ENTRY_TRAN_ADDR	TRAN_ADDR_TYPE	Transport address
CONN_ENTRY_DOMAIN_NAME	SPECT_DATA	Domain for connection
CONN_ENTRY_SERVER_NAME	SPECT_DATA	Server name for connection
CONN_ENTRY_SERVICE_NAME	SPECT_DATA	Service type name for connection
CONN_ENTRY_PERM	BIT	Permanent flag for connection
CONN_ENTRY_AUTH	BIT	Authenticated state
CONN_ENTRY_ANCHOR	BIT	Anchor state for connection
CONN_ENTRY_SUSPENDED	BIT	Suspended state for condition

scanMatchPtr Points to data that matches the data type defined by *scanInfoId* as to value to match. If *scanInfoId* defines a data member that is a pointer, then *scanMatchPtr* is a pointer to that data structure.

scanFlags Determines whether to return connection information for connections that do match the scan criteria or which do not match the scan criteria. The permitted values include:

MATCH_EQUALS	"Equal to" type lookup
MATCH_NOT_EQUALS	"Not equal to" type lookup

retInfoId Specifies which type of connection information should be returned. Acceptable values are the same as for *scanInfoId* except that the whole CONN_INFO_STRUCT can be returned (using CONN_ENTRY_RETURN_ALL) and no return information can be requested (using CONN_ENTRY_RETURN_NONE).

Supported *retInfoId* types include the following:

Value	Data type	Meaning
CONN_ENTRY_AUTH_USER_ID	UINT32	ID of user
CONN_ENTRY_AUTH_SVC_ID	UINT32	ID of authentication module
CONN_ENTRY_AUTH_HANDLE	UINT32	Authentication Handle
CONN_ENTRY_AUTH_SPEC_PTR	UINT32	Pointer to auth specific info
CONN_ENTRY_SESS_SVC_ID	UINT32	Session Protocol Provider Id
CONN_ENTRY_SESS_SPEC_PTR	UINT32	Pointer - Session specific info
CONN_ENTRY_NAME_SVC_ID	UINT32	Id of name service provider
CONN_ENTRY_MAX_IO	UINT32	Maximum IO for connection
CONN_ENTRY_MAX_RW_IO	UINT32	Maximum read/write IO
CONN_ENTRY_ROUND_TRIP	UINT32	Round trip time in milliseconds
CONN_ENTRY_SECURITY	UINT32	Security mode in effect
CONN_ENTRY_LICENSE	UINT32	License state of connection
CONN_ENTRY_TRAN_ADDR_OBJ	UINT32	Pointer to the tran addr object
CONN_ENTRY_USER_CTX_PTR	UINT32	Pointer to user context pointer
CONN_ENTRY_NCP_HOOK_RTNS	UINT32	Pointer to NCP hook routines
CONN_ENTRY_SFT_LEVEL	UINT32	Current sft level
CONN_ENTRY_TTS_LEVEL	UINT32	Current tts level
CONN_ENTRY_SERVER_CONN_NUM	UINT32	Server connection number
CONN_ENTRY_SERVER_VERSION	UINT32	Server version
CONN_ENTRY_TRAN_ADDR	TRAN_ADDR_TYPE	Transport address
CONN_ENTRY_TRAN_SVC_ID	UINT32	Transport service provider ID
CONN_ENTRY_DOMAIN_NAME	SPECT_DATA	Domain for connection
CONN_ENTRY_SERVER_NAME	SPECT_DATA	Server name for connection
CONN_ENTRY_SERVICE_NAME	SPECT_DATA	Service type name for connection
CONN_ENTRY_ERROR	BIT	Error condition of connection
CONN_ENTRY_PERM	BIT	Permanent flag for connection
CONN_ENTRY_AUTH	BIT	Authenticated state
CONN_ENTRY_ANCHOR	BIT	Anchor state for connection
CONN_ENTRY_SUSPENDED	BIT	Suspended state for condition
CONN_ENTRY_RETURN_ALL	CONN_INFO_TYPE	Structure defining all info
CONN_ENTRY_RETURN_NONE		No return Info requested

retInfoLen Length of output buffer into which to return information. This field is valid only if *retInfoId* requests a structure.

Output	*retInfoPtr*	Pointer to buffer into which to receive information. If the caller is requesting only one piece of information, then this is a pointer to a buffer of the type of information being requested. If the return type is SPECT_DATA, the size of the Data buffer must be indicated in the SPECT_DATA.*Length* field. If the return type is CONN_INFO_TYPE, all of the SPECT_DATA fields and all of the fields specifying maximum buffer sizes must be initialized before making this call. For iterative scans, this function will copy the value contained in the max name length field (that is, connMaxDomainNameLen, connMaxServerNameLen, connMaxServiceNameLen) into the appropriate *SPECT_DATA.Length* field (that is, if *connMaxServerNameLen* is set to 9, this routine will copy that value into the connServer *Data.Length* field before copying the server name. This will allow for iterative calls without the caller resetting any fields.)
	connReference	Connection reference associated with the information that is being returned. The caller can use this connection reference to open the connection and get an actual connection handle (see **CONNOpenByReference** function description) if it needs to perform any processing on this connection.

Remarks

This call scans for connections based on any piece of connection information contained in the CONN_INFO_TYPE structure. This allows the caller to look up all connection table entries matching any of the Get-/Set-Entry values in the connection table.

This lookup method can be time-consuming since the size of the connection table is not pre-determined, and the procedure must cycle through the entries one at a time while checking the appropriate information. Consequently, this procedure is designed for versatility rather than speed.

To understand how this call works, imagine that the caller wants to scan for all connections in the NDS tree "NOVELL_INC." The call would be made with the following parameters:

processGroupID = current process group id
processId = current process id
scanReference = 0 (initially)
scanInfoId = CONN_ENTRY_DOMAIN_NAME
scanMatchPtr = SPECT_DATA "NOVELL_INC"
scanFlags = MATCH_EQUALS
retInfoId = CONN_ENTRY_RETURN_NONE
retInfoLen = 0
retInfoPtr = NULL
connReference = 0

See also

CONNGetStructure
CONNGetValue
CONNSetStructure
CONNSetValue

CONNSetDefaultConnection

Description	Associates a connection handle with a process and process group.

Syntax

```
UINTXX DIST
CONNSetDefaultConnection (
    UINT32              processGroupID,
    UINT32              processId,
    CONN_HANDLE connHandle)
```

Input

processGroupID	Calling function's process group ID.
processID	Calling function's process ID.
connHandle	The connection handle to associate with the specified process identifiers.

Output None.

See also CONNGetDefaultConnection

CONNSetPassword

Description

Synchronizes a password change across a domain (several bindery servers, and/or several trees). The caller specifies whether a dialog box requests the old and new passwords (allowing for greater security to be built into applications).

Syntax

```
UINTXX
CONNSetPassword
    AUTH_HANDLE   authHandle,
    UINT32                  flags,
    SPECT_DATA DIST *password)
```

Input

authHandle	Authentication handle to set the password for.
flags	Controls whether a secure prompting for the password is made from ring 0. The flags may have one of the following values: CONN_PASSWD_PROMPT_NONE CONN_PASSWD_PROMPT
password	Password, stored in SPECT_DATA structure. It must be correctly initialized. If the password is to be prompted for from ring 0, this parameter should be set to NULL.

Output

None.

See also

CONNAuthenticateWithHandle
CONNAuthenticate
CONNUnauthenticate
CONNCreateAuthenticationHandle
CONNDestroyAuthenticationHandle
CONNScanAuthenticationHandles
CONNGetAuthHandleInfo
CONNVerifyPassword

CONNSetStructure

Description

Sets a specific connection structure for the given connection handle.

Syntax

```
UINTXX DIST
CONNSetStructure (
    CONN_HANDLE connHandle,
    UINT32          infoId,
    UINT32          infoLen,
    VOID DIST      *infoPtr)
```

Input

connHandle Connection Handle.

infoId Connection parameter, which can be one of the following:

Value	Data type	Meaning
CONN_ENTRY_TRAN_ADDR	TRAN_ADDR_TYPE	Transport address
CONN_ENTRY_DOMAIN_NAME	SPECT_DATA	Connection's Domain name
CONN_ENTRY_SERVER_NAME	SPECT_DATA	Connection's Server name
CONN_ENTRY_SERVICE_NAME	SPECT_DATA	Connection's Service name

Note: *These structures should only be set by client internal NLMs!*

infoLen Length of input buffer from which to take information. If *infoId* is a SPECT_DATA structure, *infoLen* should be the size of the SPECT_DATA structure and the *Data* field of the SPECT_DATA structure should point to a valid Data string. The *Length* field of the SPECT_DATA structure should accurately indicate the length of the *Data* field of that structure. (See the example listed with **CONNGetStructure**.)

If the *infoId* is CONN_ENTRY_RETURN_ALL, the *infoLen* parameter should be the size of the CONN_INFO_STRUCT. All SPECT_DATA Data pointers should be initialized to a valid *Data* string

	and all SPECT_DATA lengths should be initialized to the length of the buffer associated with the SPECT_DATA *Data* pointer.
infoPtr	Pointer to the buffer from which to set information into the *connEntry*.

Remarks

Note: *This call should be used only by CLIENT INTERNAL NLMs!!*

See also

CONNGetStructure
CONNGetValue
CONNSetValue
CONNScanInfo

CONNSetValue

Description

Sets specific connection entry information for the given connection handle.

Syntax

```
UINTXX DIST
CONNSetValue (
    CONN_HANDLE connHandle,
    UINT32          infoId,
    UINT32          infoValue)
```

Input

connHandle Connection handle for which the value should be set.

infoId Connection parameter, which can be one of the following:

Avail	Value	Datatype	Meaning
I	CONN_ENTRY_AUTH_USER_ID	UINT32	ID of user
I	CONN_ENTRY_AUTH_SVC_ID	UINT32	ID of authentication module:
			AUTH_SVC_BINDERY_ID
			AUTH_SVC_NDS_ID
			AUTH_SVC_PNW_ID
I	CONN_ENTRY_AUTH_HANDLE	UINT32	Authentication Handle
I	CONN_ENTRY_AUTH_SPEC_PTR	UINT32	Ptr to auth-specific information
I	CONN_ENTRY_SESS_SVC_ID	UINT32	Session Protocol Provider ID:
			NCP_SESSION_ID
			SMB_SESSION_ID
I	CONN_ENTRY_SESS_SPEC_PTR	UINT32	Ptr to session-specific information
I	CONN_ENTRY_NAME_SVC_ID	UINT32	ID of name service provider:
			NAME_SVC_BINDERY_ID
			NAME_SVC_NDS_ID
			NAME_SVC_PNW_ID
I	CONN_ENTRY_MAX_IO	UINT32	Maximum IO for connection
I	CONN_ENTRY_MAX_RW_IO	UINT32	Maximum read/write IO
I	CONN_ENTRY_ROUND_TRIP	UINT32	Round trip time in milliseconds
I	CONN_ENTRY_SECURITY	UINT32	Security mode in effect
			Bit definitions:
			CFG_CRC
			CFG_MD4
			CFG_CRYPT
I	CONN_ENTRY_LICENSE	UINT32	License state of connection

```
I  CONN_ENTRY_TRAN_ADDR_OBJ   UINT32      Pointer to the tran addr object
I  CONN_ENTRY_NCP_HOOK_RTNS   UINT32      Pointer to NCP hook routines
I  CONN_ENTRY_SFT_LEVEL       UINT32      Current sft level
I  CONN_ENTRY_TTS_LEVEL       UINT32      Current tts level
```

Avail	Value (continued)	Datatype	Meaning
I	CONN_ENTRY_SERVER_CONN_NUM	UINT32	Server connection number
I	CONN_ENTRY_SERVER_VERSION	UINT32	Server version
I	CONN_ENTRY_PERM	BIT	Permanent flag for connection
I	CONN_ENTRY_AUTH	BIT	Authenticated state
I	CONN_ENTRY_ANCHOR	BIT	Anchor state for connection
I	CONN_ENTRY_SUSPENDED	BIT	Suspended state for condition
I	CONN_ENTRY_ORDER_NUM	UINT32	Session connection order
I	CONN_ENTRY_ORDER_NUM	UINT32	Connection order number

The availability of these items is indicated in the first column.
A Available to all calling functions.
I Available to internal client NLMs only.

infoValue The data value to set. All bit values are zero to clear; any other value will set the bit.

See also

CONNGetStructure
CONNGetValue
CONNSetStructure
CONNScanInfo

CONNUnauthenticate

Description

Unauthenticates a connection handle by calling down to the authentication multiplexor. If the connection is not already authenticated, an error will be returned. The correct authentication handle is determined by interrogating the *connHandle* for the information.

Syntax

```
UINTXX DIST
CONNUnauthenticate (
    CONN_HANDLE connHandle)
```

Input

connHandle	The connection to unauthenticate.

Output

None.

See also

CONNAuthenticateWithHandle
CONNAuthenticate
CONNCreateAuthenticationHandle
CONNDestroyAuthenticationHandle
CONNScanAuthenticationHandles
CONNGetAuthHandleInfo
CONNChangePassword
CONNVerifyPassword

CONNValidateHandle

Description

Checks the validity of a connection.

Syntax

```
UINTXX DIST
CONNValidateHandle (
    CONN_HANDLE connHandle,
    UINT32             flags)
```

Input

connHandle The connection of interest.

flags Controls the type of validation performed on the connection. The permitted values include:

```
CONN_VALIDATE_HANDLE
         Verify only that connHandle is
         valid.
CONN_VALIDATE_SESSION
         Verify through to the far end.
```

Output

None.

Remarks

ConnMan will check the validity of the connection at its level (that is, see that the *connHandle* is valid) and, if so, will call the session protocol associated with the connection using **SESSValidateConnection** and verify the connection.

See also

None.

CONNVerifyPassword

Description

Verifies a password for a given domain (consisting of several bindery servers, and/or several trees).

Syntax

```
UINTXX DIST
CONNVerifyPassword (
    UINT32              domainHandle,
    UINT32              flags,
    SPECT_DATA DIST *password)
```

Input

authHandle	Authentication handle for which to set the password.
flags	Controls whether a secure prompting for the password is made from ring 0. The flags may have one of the following values: CONN_PASSWD_PROMPT_NONE CONN_PASSWD_PROMPT
password	Password, stored in a SPECT_DATA structure. It must be correctly initialized. If the password is to be prompted for from ring 0, this parameter should be set to NULL.

Output

None.

Remarks

The *flags* parameter allows the caller to specify whether this function should put up a dialog box requesting the password to verify (allowing for greater security to be built into applications).

See also

CONNAuthenticateWithHandle
CONNAuthenticate
CONNUnauthenticate
CONNCreateAuthenticationHandle
CONNDestroyAuthenticationHandle
CONNScanAuthenticationHandles
CONNGetAuthHandleInfo