

YNQ User's Guide

YNQ 1.6.0

Document version: 0.51

Table of Contents

1	REFERENCED DOCUMENTS	4
2	INTRODUCTION.....	5
3	END USER GUIDE TO YNQ SMB SERVER.....	6
3.1	BROWSING FILES AND DIRECTORIES	6
3.1.1	<i>Files.....</i>	6
3.1.2	<i>Directories</i>	6
3.2	SYNCHRONIZING FOLDERS	6
3.3	MANAGING SHARED FOLDERS.....	7
3.3.1	<i>List of Shares.....</i>	7
3.3.2	<i>Extended Services</i>	7
3.3.3	<i>Access Rights</i>	8
3.4	LOCAL USER MANAGEMENT	10
4	PROGRAMMERS GUIDE TO YNQ API	11
4.1	PORTING AND INTEGRATION	11
4.2	YNQ ARCHITECTURE	11
4.3	UNICODE SUPPORT	12
4.3.1	<i>System Interface.....</i>	12
4.3.2	<i>Application Interface</i>	12
4.4	YNQ API STRUCTURE.....	12
4.4.1	<i>API Naming Convention</i>	12
4.4.2	<i>API Groups</i>	13
4.5	DNS AND NETBIOS	13
4.5.1	<i>The Resolver.....</i>	13
4.5.2	<i>DNS and NetBIOS limitations.....</i>	13
4.6	YNQ CLIENT	14
4.6.1	<i>Memory Consumption.....</i>	14
4.6.2	<i>Error Reporting</i>	15
4.6.3	<i>DFS support.....</i>	15
4.6.4	<i>Message Signing</i>	15
4.6.5	<i>SMB1 Support.....</i>	16
4.6.6	<i>SMB2 Support.....</i>	16
4.6.7	<i>SMB3/SMB311 Support and Encryption</i>	16
4.6.8	<i>Domain Controller Name Resolution</i>	16
4.6.9	<i>Network Discovery.....</i>	16
4.6.10	<i>Domain Membership</i>	17
4.6.11	<i>LDAP</i>	17
4.6.12	<i>WS-Discovery</i>	17
4.6.13	<i>Authentication Types</i>	17
4.6.14	<i>Synchronous vs. Asynchronous Operations.....</i>	18
4.7	YNQ SERVER	18
4.7.1	<i>YNQ Server API versus YNQ Server Control API.....</i>	18
4.7.2	<i>SMB1 Support.....</i>	18
4.7.3	<i>SMB2 Support.....</i>	18
4.7.4	<i>SMB3/SMB311 Support and Encryption</i>	18
4.7.5	<i>Domain Membership and Pass-through Authentication.....</i>	19

4.7.6	<i>Message Signing</i>	<i>19</i>
4.7.7	<i>Handling Client Connections.....</i>	<i>20</i>
FIGURE 1 DEFAULT SECURITY DESCRIPTOR.....		9

1 Referenced Documents

- [1] NQ Server Integration and Porting Guide
- [2] NQ Client Integration and Porting Guide
- [3] NQ Library Reference folder
- [4] Common Internet File System (SMB). Technical Reference. Revision: 1.0.
SNIA

2 Introduction

This document achieves two different goals.

First, it explains how to work with files and directories on a remote YNQ SMB Server (see 3). Instead of providing a complete user manual this document rather concentrates on the differences between using YNQ based servers and Windows based servers. This part of the document is intended for end-users and system administrators.

The second aim of this document is to explain the structure of YNQ software and how to use its API (see 4). This part of the document is intended for software developers implementing YNQ software into their projects. Processes, described below assume that Porting and Integration process has been completed. See [1] for information about YNQ Porting and Integration.

3 End User Guide to YNQ SMB Server

Using NQ-based SMB Server is very much similar to using a Windows-based Server. A user can access an NQ-based file server from his desktop computer using any of its tools: Windows Explorer, MS Word, Linux shell, Mac OS shell, etc. However, using YNQ-based server might introduce some minor differences. Most of them are a result of limitations applied by the underlying file system.

Further in this section we use Windows Explorer's behavior as an etalon and we will be mostly comparing describing YNQ SMB Server's behavior as observed over Windows Explorer. Cases applicable to another client tool will be explicitly specified.

3.1 *Browsing Files and Directories*

3.1.1 Files

- File names:** On some operating systems file names may contain characters not allowed by Windows.
- File times:** File "accessed", "created" and "modified" times may be not accurate due to time-zone difference between client and server computers. This happens when the underlying operating systems on an NQ-based server does not report proper time zone.
- Read-only:** As YNQ implementations depend of the underlying file system, some implementations may allow deleting and modifying read-only files.
- File attributes:** Some very unusual file systems do not keep "read-only" and/or "system" and/or "hidden" attributes.

3.1.2 Directories

- Directory names:** On some operating systems directory names may contain characters not allowed by Windows.
- Read-only:** Latest Windows versions treat read-only folder status as an indication only, allowing creating, modifying and deleting files and folders in a read-only directory. In an NQ-based server this behavior depends on the underlying file system. The default YNQ implementation for VxWorks, Linux and Windows CE copies the Windows approach to read-only folders. However, some YNQ implementation may deviate from this behavior.
- Directory attributes:** Some very unusual file systems do not keep "read-only" and/or "system" and/or "hidden" attributes.

3.2 *Synchronizing Folders*

Two SMB clients connected to the same YNQ Server are always synchronized. For instance, if a file is created on YNQ Server through Windows Explorer on one client it will appear in a Windows Explorer window on another client providing that both clients are exploring the same server folder.

This synchronization is limited to SMB traffic only. For instance, if a file was added locally or through, say, FTP, the synchronization will not happen automatically. To be synchronized with non-SMB modifications a user may use Refresh command (e.g., -F5) to bring his view into a sync with the server's contents.

3.3 Managing Shared Folders

YNQ SMB Server grants three levels of Remote Management for shared folders. The basic level of support is included with UD_CS_INCLUDERPC_SRV SVC compile-time parameter (see [1]). This level allows only viewing the list of shares and share properties. When UD_CS_INCLUDERPC_SRV SVC_EXTENSION (see [1]) is also defined, YNQ SMB Server features Extended Services (see 3.3.2). And finally, with UD_CS_INCLUDESECURITYDESCRIPTORS (see [1]) YNQ SMB Server is capable of defining voluntary access rights over Remote Management (see 3.3.3).

3.3.1 List of Shares

A list of shares from YNQ SMB Server when shown over Remote Management on a Windows client specifies share paths with a drive-letter prefix. Drive letters correspond to so-called “hidden” shares on YNQ Server (see *udGetNetShare* function in [1]). Drive letters represent one or more file system roots on the target machine. For example, assume that one of the hidden shares is named C\$ and specifies a root path “/a0ta0a” on a VxWorks target. Then, a share mapped on “/ata0a/share1” will be displayed in the Remote Manager as having path “C:/share1”.

When no hidden share is defined on YNQ Server, a share will be always displayed with “C:” prefix for compatibility with MS Windows. For instance, the same share as above will be displayed with path “C:/ata0a/share1”.

Any user is capable of viewing shares.

3.3.2 Extended Services

This level of support features the following:

- Adding/removing shares
- Viewing a list of connected users
- Disconnecting a user
- Viewing a list of open files
- Closing a file

When UD_CS_INCLUDESECURITYDESCRIPTORS (see [1]) is defined these services are available only for a user with Administrator rights. Such user may be either local Administrators or network Administrator. When UD_CS_INCLUDESECURITYDESCRIPTORS (see [1]) is not defined, any user can benefit from Extended Services.

Adding shares can be performed in one of two ways:

1. Using Browse button in Add Share Wizard, user then chooses desired folder under one of drive letters (usually - C\$) - this functionality requires defining at least one hidden share, see section 8.2.2.11 in [1] . A hidden share cannot be removed. When defining hidden share and user shares best practice is to specify all local paths using either absolute or relative paths, but not the mix of both. For example if hidden share has mapping path of relative form and one of user shares has mapping path of absolute form, such user share will be displayed in Share Management having empty path (share functionality is not affected).
2. Specifying local path in Add Share Wizard, path should be of form “C:/ata0a/folder” (VxWorks), this functionality applies when no default administrative share is defined.

3.3.3 Access Rights

YNQ Software supports share-level access rights. This is achieved by means of Security Descriptors.

3.3.3.1 Managing Security Descriptors

YNQ Software supports Security Descriptors for shares. This functionality is controlled by the `UD_CS_INCLUDESECURITYDESCRIPTORS` compile-time parameter (see [1]). With Security Descriptors, YNQ SMB Server is capable of:

- Setting up a read-only share.
- Setting up a hidden share.
- Restricting access to a share for a particular user or a particular user group.

There are two methods of managing Security Descriptors: 1) Programmatically and 2) Remotely. *Programmatically* managing Security Descriptors requires application support. It is based on persistent Security Descriptors (see 3.3.3.2) and it assumes that user application modifies Security Descriptors out of the YNQ scope. See also [1] for using persistent storage for Security Descriptors. A Windows user can *remotely* manage Security Descriptors on an YNQ SMB Server host by means of Windows Computer Management tool.

NOTE: Sometimes Windows does not allow opening the list of shares. Error message reads: “The following error occurred while reading the list of shares for Windows clients: Error 5: Access is denied.” This happens because Windows opens new connection to the computer to be managed. If this connection has insufficient privileges, it fails and the current context does not allow explicitly changing the credentials. There is a workaround for this problem. Before entering Computer Management do the following:

1. Choose Run in the Start menu. In the popped up text box type network path to the target computer to be managed as: `\\<hostname>`
2. Click OK. Windows Explorer will show the list of shares.
3. Open any of the shares. If your credentials are not sufficient, Windows will prompt you for new credentials.
4. Continue to Computer Management.

In this case, Windows shares the same connection that was established during the workaround steps, where new credentials were explicitly set.

The following restrictions apply:

1. YNQ Software manages share access rights through the Share Permissions tab. Security tab is not supported.
2. YNQ Software features two Locations: 1) local host and 2) domain. For the support of the latter YNQ Software should be a member of domain.

3.3.3.2 Persistent and Default Security Descriptors

A *Persistent* Security Descriptor is a descriptor that keeps its value between two subsequent YNQ Server starts. When YNQ starts up it attempts to load share Security Descriptors from user application by means of `usLoadShareSecurityDescriptor()` calls (see [1]). When share descriptor is modified (either remotely by means of Computer Management or programmatically) YNQ attempts to save this descriptor by calling `usSaveShareSecurityDescriptor()` (see [1]). Both load and save operations are out of the YNQ scope and should be provided by the application software. In this case descriptors are stored between YNQ starts. When `usLoadShareSecurityDescriptor()`

call fails the appropriate descriptor is not persistent anymore and YNQ loads a *default* Security Descriptor instead. The default descriptor has the following form:

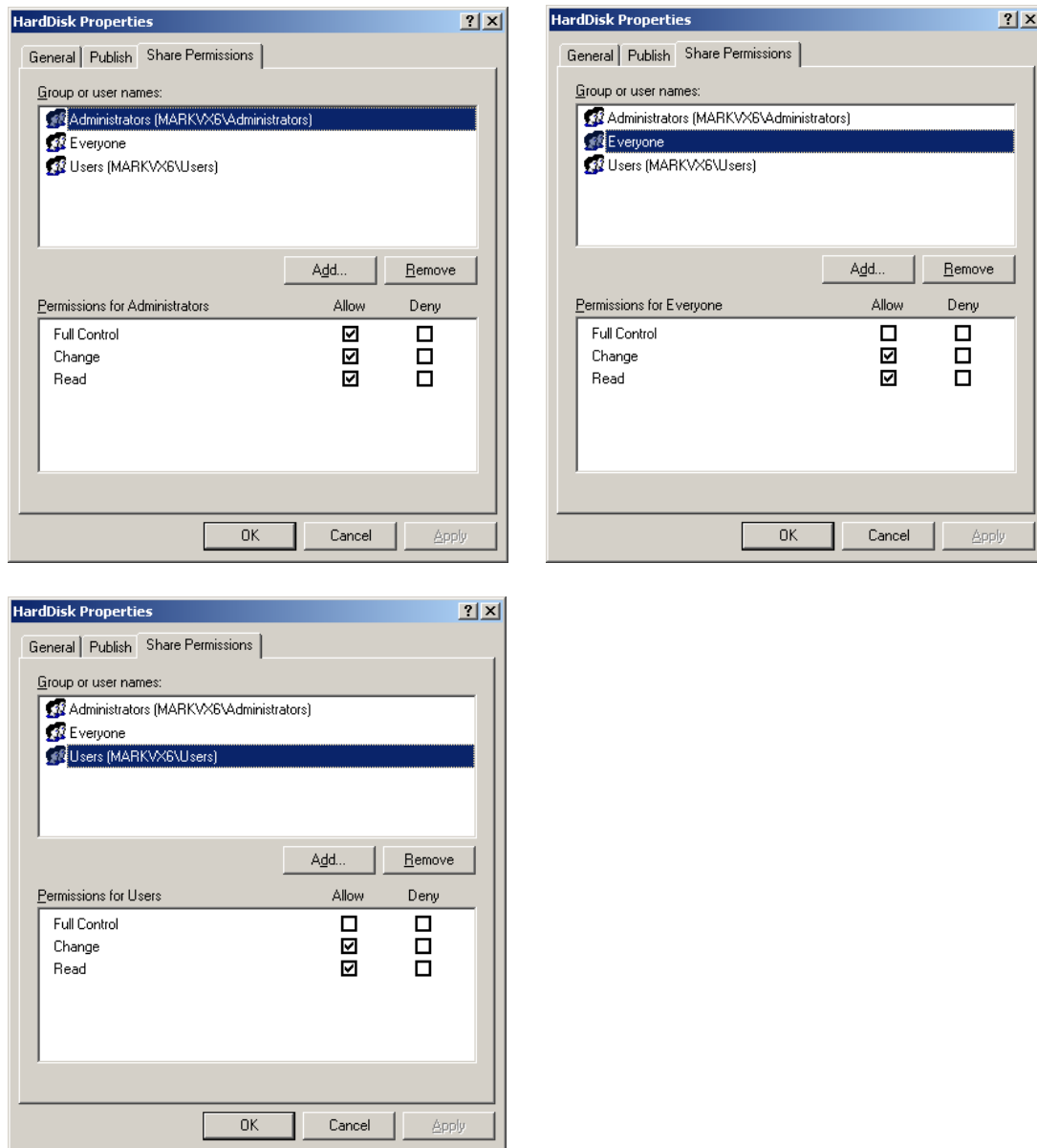


Figure 1 Default Security Descriptor

If descriptors are not persistent YNQ SMB Server will start again with a default descriptor applied to each share. With persistent descriptors this will happen on the first run when descriptor was not stored yet or after a stored descriptor got corrupted or removed.

3.3.3.3 User Authentication and Security Descriptors

The correspondence between users on the YNQ SMB Server and on a client machine depends on the authentication method. YNQ SMB Server supports three means of authentication:

- **No Authentication.** With this authentication method YNQ Server does not distinguish between users. There are no Administrators as well, so shares' Security Descriptors may be managed only programmatically. A user,

connected to such YNQ SMB Server will match “Everyone” in the security descriptor.

- **Local Authentication.** The result of this authentication depends on application’s authentication mechanism provided during integration (see [1]). In this case YNQ Server distinguishes between individual local users as well as between two built-in groups: Users and Administrators. For instance, a share may be defined read-only for all users except Administrators.
- **Pass-Through Authentication.** With this method enabled, YNQ SMB Server is capable of delegating the authentication process to a Domain Controller. This method allows maximum remote share management with access rights on a per-user basis.

See [1] for more information for how to provide user authentication for YNQ SMB Server.

3.4 Local User Management

YNQ Server supports Local Users and Groups sub-tree of Computer Management. The following restrictions apply:

- YNQ Server has two built-in groups: Users and Administrators. A new group cannot be created and neither of two built-in groups can be removed.
- No policy is applicable to passwords. A password can be any length.
- YNQ Server does not add pre-defined users. Instead it reports only those users that are provided by the project layer.
- The persistent mechanism for user’s name, password, full name and description is out of the YNQ scope. For instance, if the project layer saves user name and password but do not save full name and description, the latter two would not be set remotely. See [1] for saving and retrieving user information.

4 Programmers Guide to YNQ API

4.1 Porting and Integration

Prior to developing a NQ-based application user must accomplish porting and integration. Document [1] describes them in details. Note, that when YNQ software is shipped, it is usually ported to the target operating system. This porting, however, concerns the mostly typical hardware/firmware/software configurations for the given OS. Therefore, a user may need to convey a partial porting process in order for fine tune YNQ for his target platform.

4.2 YNQ Architecture

YNQ software consists of the following main modules:

- **NetBIOS Daemon** (also referred to as **nd**) - resident service listening on the NetBIOS Name Service UDP port (137) and NetBIOS Datagram Distribution port (138) and responsible for NetBIOS name resolution and broadcast datagram distribution.
- **SMB Server Daemon** (also referred to as **cs**) – resident service listening on the NetBIOS Session Service TCP legacy port (139) and/or on Naked SMB TCP port 445 and responsible for SMB file and printer sharing.
- **SMB Client Library** (also referred to as **cc**) – a library which provides a SMB remote file and printer access as well as the Browser services to an application it is linked to. For some operating systems Visuality provides encapsulation of SMB Client library into a file system driver, which allows user applications to call standard file I/O functions (open(), read(), write(), close(), etc.) to access remote shared files.
- **LDAP Library** (also referenced as **ld**) – an abstraction API for accessing Active Directory over the LDAP protocol. YNQ Client does not contain its own LDAP client but rather implements a wrapper over an existing LDAP client, whose implementation is out of YNQ scope.

Additionally, YNQ has the following supplementary modules:

- **Common Library** (also referred to as **cm**) – a library of common functions used by other modules in the YNQ system independent core.
- **Authentication Library** (also referred to as **am**) – a library of authentication functions used by other modules in the YNQ system independent core.
- **NetBIOS Socket Library** (also referred to as **ns**) – a library providing NetBIOS/IPv4/IPv6 Socket API to SMB Server/Client and Browser modules.
- **System Abstraction Library** (also referred to as **sy**) – a library providing the operating system abstraction layer. The code of this module is operating system dependent.
- **User Defined Library** (also referred to as **ud**) – a set of user defined callback functions used mostly for YNQ configuration as well as YNQ compile-time parameters. The code of this module is operating system dependent.
- **Start Module** (also referred to as **st**) – module providing YNQ start/stop API. The code of this module is operating system dependent.
- **File System Driver** (also referred to as **fs**) – SMB Client file system driver for SMB Client encapsulation. This module is optional and implemented for part of the operating systems. The code of this module is operating system dependent.

4.3 Unicode Support

The Unicode support in YNQ is separated into two levels:

- System interface
- Application interface

Terminology note: The term “Unicode” applies to UCS-2LE. YNQ is using LE Unicode string regardless of the target platform.

4.3.1 System Interface

On the System level YNQ Supports Unicode system API. All the internal string types are defined as ***NQ_WCHAR***. Refer to [1] for more information.

4.3.2 Application Interface

Unlike System level the Application level provides both compile-time and run-time Unicode support. Every Unicode capable YNQ API function has three interfaces:

- ASCII enforced – the interfaces of this type use string types defined as ***NQ_CHAR (char)*** and their names are postfix-ed with ‘A’. Example: ***ccCreateFileA()***.
- Unicode enforced (default) – the interfaces of this type use string types defined as ***NQ_WCHAR (short)***. Example: ***ccCreateFile()***.

NQ Library Reference [3] addresses only the default functions of every Unicode-capable triplet, however if a particular function has at least one of its parameters defined as string then it can be called in either of three ways explained above.

Additionally, NQ Library Reference defines a number of Unicode dependent data structures. If a particular data structure has at least one of its fields defined as string then there are two additional data structures: one for ASCII calls, postfix-ed with ‘A’. For example, when calling ***ccFindFirstFileA()***, the `findFileData` parameter should be defined as `FindFileDataA_t`, while for ***ccFindFirstFile()***, it should be defined as `CCFindFileData`.

4.4 YNQ API Structure

This section explains the functional structure of YNQ API functions.

4.4.1 API Naming Convention

Every API function name is prefixed with either module abbreviation or with “***nq***”.

The ***YNQ*** prefix is used for those API functions which are exposed to an application in cases when YNQ is most completely ported and integrated into the target operating system. When part of the YNQ functionality is not required or impossible to integrate, then application can call module specific calls, which names are prefix-ed with module abbreviation.

Example: SMB Client being completely integrated into the target OS is encapsulated into a file systems driver and application calls standard file I/O functions (***open()***, ***read()***, etc.). If the file system driver is not implemented then application can call SMB Client API directly (***ccCreateFile()***, ***ccReadFile()***, etc.).

4.4.2 API Groups

NQ Library Reference [3] document has a functional summary table which has all the API calls grouped by their corresponding module.

4.5 *DNS and NetBIOS*

4.5.1 The Resolver

In YNQ the name “Resolver” stands for a mechanism responsible for the following:

- Resolving IP address/addresses by host name. This method is further mentioned as “resolution”.
- Resolving host name by IP address. This method is further mentioned as “reverse resolution”.

YNQ features several resolution methods:

1. DNS query.
2. LLMNR query.
3. NetBIOS unicast query.
4. NetBIOS local broadcast query.

All the above methods use datagram queries. Methods 1 and 3 are unicast methods, while methods 2 and 4 are multicast methods. YNQ allows defining multiple DNS servers and multiple WINS servers (for NetBIOS queries).

YNQ Resolver conveys resolution in two steps:

1. It transmits all available unicast queries concurrently. This includes queries to each available DNS server and queries to each available WINS. If at least one of the unicast queries succeeds, Resolver does not perform the second step. When performing a resolution, Resolver successfully quits after all responses are either received or timed out. The resulted list of IP addresses is a merge of IP addresses from each successful response. When performing a reverse resolution, Resolver successfully quits after receiving the first successful response. Subsequent responses become discarded.
2. This step is only performed if all the methods on the first step has either timed out or received errors. Then, Resolver concurrently all available transmits multicast queries. This includes LLMNR queries and NetBIOS broadcasts. YNQ treats query results on this step exactly in the way as in step 1.

4.5.2 DNS and NetBIOS limitations

The following limitations are applied:

- YNQ does not detect duplicate names. When it receives two responses with different IP addresses, it treats them as two IPs of the same host. On connection, YNQ Client enumerates them one by one until connected so that the first received address will be the first to try.
- When YNQ Server performs startup it may receive NAME_IN_CONFLICT NetBIOS response. In this case YNQ Server fails to start up. YNQ does not specifically detect conflict been solved. You may re-run YNQ Server later after the conflict was solved.

4.6 YNQ Client

4.6.1 Memory Consumption

YNQ Client implicitly allocates memory blocks. The table below designates maximum memory consumption per particular objects.

Object	Released Explicitly	Can be shared	Max consumption	Max name concerned
Mount point	Yes		600	Yes
Server		Yes	1800	Yes
User		Yes	1200	Yes
Share		Yes	700	Yes
File	Yes		650	Yes
Search			600	Yes
DFS cache		Yes	1200 per entry	Yes
Resolver cache		Yes	550 per entry	Yes

Notes: The amount memory allocated differs by API calls history and call parameters. It also depends on the target platform architecture and compiler. Therefore, the numbers above are approximate. In most cases real numbers will be significantly less. Following are the main factors of the memory consumption:

- The numbers above are approximate.
- The numbers above reflect a 64-bit architecture. On a 32-bit architecture memory consumption may be smaller.
- Size of some component (mutexes, semaphores, etc.) depends on the target platform. The numbers above reflect maximum known size.
- Some objects may be shared between API calls. For instance, two `nqAddMount()` calls can share the same Server object.
- A value “Yes” in the column “Max name concerned” means that the memory consumption number reflects a maximum of 256 UTF16 characters in the name. Real consumption is much less.
- In Debug mode memory consumption is bigger.
- The list above does not reflect memory blocks that are allocated and destroyed in the context of the same API call, like request and response buffers, temporary names, etc.

The following memory blocks were considered:

Mount point: An object of this kind is explicitly allocated on `nqAddMount()` call and it is explicitly released on `nqRemoveMount()` call.

Server: An object of this kind is implicitly allocated per a server connection. It is implicitly released on server disconnect. Server objects may reflect mount points, may be shared between mount points and also can be implicitly created during DFS resolving.

User: An object of this kind is implicitly allocated per a user connection. It is implicitly released on server disconnect. User objects may reflect mount

points, may be shared between mount points and also can be implicitly created during DFS resolving.

Share: An object of this kind is implicitly allocated per a tree connection. It is implicitly released on tree disconnect. Share objects may reflect mount points, may be shared between mount points and also can be implicitly created during DFS resolving.

File: File objects are explicitly created on `ccCreateFile()` call and they are explicitly destroyed on `ccCloseHandle` call. Also, File objects are implicitly created and destroyed on RPC calls as well as on many SMB2 information queries.

Search: Such objects are allocated on `ccFindFirstFile()` call and they are released when search is over or on `ccFindClose()` call.

DFS cache: Entries in DFS cache are implicitly created during DFS resolution. They are released either when their TTL expires or on YNQ shutdown.

Resolver cache: Entries in Resolver cache are implicitly created during DFS resolution. They are released either when their TTL expires or on YNQ shutdown.

4.6.2 Error Reporting

YNQ Client uses the OS native mechanism for error reporting. For example, most of the POSIX compliant applications should examine *errno* variable for the failure reason, while YNQ Client is responsible for setting this value. See document [3] for YNQ error codes.

NOTE: YNQ sets the most significant 16 bits of the error codes to 0x00FF to distinguish them from standard POSIX codes.

4.6.3 DFS support

YNQ SMB Client is capable of resolving a DFS path. This functionality is included with `UD_NQ_INCLUDEDDFS` parameter (see [1]). DFS resolving is fully transparent from the YNQ API point of view so that any path in a mount or file operation might be either a DFS path or a regular path. YNQ Client attempts DFS resolving first and then, if this resolving fails, it tries a regular path. When DFS resolving is turned on, calling those API functions that involve a file/mount paths might introduce some additional delays.

4.6.4 Message Signing

YNQ SMB Client supports SMB/SMB2 message signing. This functionality is optional and the decision whether to sign SMB/SMB2 traffic is controlled by a conjunction of two factors.

The first factor is a message signing flag inside YNQ Client. This value is initially set to `FALSE` and a user can change it by specifying the `in` parameter of the `ccSetSigningPolicy()` function (see [3]). The value of this parameter specifies user's preference to whether signing (`TRUE`) or not signing the traffic.

The second factor is the message signing policy on the server. It may take one of three values: "not supported", "supported", and "required". Actual message signing is the following production of the two factors:

YNQ signing State	Server signing policy	Traffic is signed
----------------------	-----------------------	-------------------

TRUE	NOT SUPPORTED	No
TRUE	SUPPORTED	Yes
TRUE	REQUIRED	Yes
FALSE	NOT SUPPORTED	No
FALSE	SUPPORTED	No
FALSE	REQUIRED	Yes

4.6.5 SMB1 Support

SMB1 support can be disabled in YNQ Client, as considered as less secured over recent years. Please note that this disabled also BROWSER protocol and RAP transactions performed over SMB1 only, affects Network Discovery.

4.6.6 SMB2 Support

SMB2 support has become the default dialect for communication, can be disabled (leading to SMB3 and SMB311 being disabled), not recommended, since SMB1 only protocol can be not used by many Windows Servers due to security issues.

4.6.7 SMB3/SMB311 Support and Encryption

SMB3 and SMB311 support is optional and it requires more resources (see [1]). YNQ Client presents the list of supported dialects during Negotiate phase and Server chooses the max dialect supported. If Server requires encrypted SMB3/SMB311 communication (global or per specific share) YNQ Client sends encrypted data.

4.6.8 Domain Controller Name Resolution

YNQ Client may need to discover the IP address of the Domain Controller. YNQ performs this as an internal operation during one of the following transactions:

- Discovering domains during Network Discovery (see 4.6.9).
- Joining domain (see 4.6.10).
- Domain logon (see 4.6.10).

YNQ resolves DC name using a combination of two protocols. Initially it attempts to resolve DC by means of SMB NETLOGON protocol by sending UDP datagrams to the Domain Master Browser. This method is only available when YNQ is generated with NetBIOS support. Even when NetBIOS is available some Windows servers fail to respond to this protocol. In this case YNQ falls back to using DNS. To support this option, DNS server should be configured to contain a record for a special name: `_ldap._tcp.dc._msdcs.<domain_name>`. See <http://support.microsoft.com/kb/816587> for more information.

4.6.9 Network Discovery

YNQ Client is capable of network discovery. An application benefits from this functionality by calling YNQ API (see [3]). The following steps are distinguished:

1. Discovering domains
2. Discovering hosts in a particular domain
3. Discovering shares on a particular host

An NQ-capable application can start from any step. For instance, if the target domain is not known yet, the application can discover known domains. However, if the domain name is preset, the application can start directly from step 2.

The most complicated step is domain discovery. To discover domains YNQ uses a combination of two methods:

1. Withdrawing Domain Backup list. This is only available with NetBIOS protocol enabled.
2. Querying Domain Controller for Trusted Domains.

The first method grants better accuracy, while the second one requires Trusted Domain configuration to be manually set into Active Directory. However, the first method is only available when YNQ is compiled with NetBIOS support.

With the addition of WS-Discovery client YNQ can discover devices on the subnet regardless Domain belonging.

4.6.10 Domain Membership

This functionality is essential for accessing Active Directory. It may be also used for other domain-member activities that might be out of the YNQ scope. The following transactions are available:

1. Joining domain. This one-time transaction is performed when an NQ-capable device is first installed into the network. It results in getting computer “secret” - a blob to be used later for domain logon. Application should persistently keep this blob.
2. Domain logon. This transaction is performed once on YNQ startup. It uses the result of the previous transaction to log the target computer in the domain.
3. Leaving domain. This is also a one-time operation performed when an NQ-capable device is removed from a network.

Domain membership functionality supports NetBIOS computer names with maximum length 15 characters. See <http://support.microsoft.com/kb/909264>

4.6.11 LDAP

YNQ Client uses LDAP open library to perform queries on LDAP Database of Active Directory, also has the ability to perform transaction of publishing printer on Active Directory.

4.6.12 WS-Discovery

YNQ Client uses WS-Discovery protocol to discover devices on the network, used in [Network Discovery](#) section.

4.6.13 Authentication Types

YNQ Client supports the following types of authentication (in low to high order):

1. LM and NTLM encryption.
2. NTLM only encryption.
3. NTLMv2 encryption.
4. GSSAPI exchange. This method supports two security mechanisms:
 - a. NTLMSSP with NTLMv2 encryption inside.
 - b. Kerberos.

Note: For Kerberos exchange YNQ uses external Kerberos implementation as specified in [1].

Higher types of authentication grant better security. YNQ Client API allows specifying the highest authentication method as available through the `ccSetMinSecurityLevel()` and `ccSetMaxSecurityLevel()` call (see [3]). When YNQ Client fails to authenticate with the highest available method, it descends to a lower one. This is repeated until a successful authentication. The highest authentication method remains reduced until either

application calls *ccSetMaxSecurityLevel()* with a higher authentication level or until YNQ Client restarts.

4.6.14 Synchronous vs. Asynchronous Operations

YNQ Client may perform read and write operations in one of two modes: 1) synchronous or 2) asynchronous.

When performing a synchronous operation, YNQ API call returns only after the entire operation was completed. For instance, when reading into a very big buffer, YNQ may issue several concurrent Read requests. However, if this operation is synchronous, it will wait for all Read requests before returning control to the caller.

When YNQ performs an asynchronous operation, it will issue all necessary requests and it will return control to the caller without waiting for responses. YNQ will then process responses concurrently. When all responses will come back, YNQ will call a callback function specified by caller.

4.7 YNQ Server

4.7.1 YNQ Server API versus YNQ Server Control API

YNQ Server control includes several operations such as stopping the server, adding/removing shares, etc. In previous YNQ versions these operations were only available in the server address space. In NQ 6.00 YNQ Server Control API was introduced. This API is available from other application's address space. This method is preferable while YNQ still supports old single-address-space calls for backwards compatibility. The names of the new YNQ Server Control API functions are prefixed with *csCtrl*. See [3] for details.

4.7.2 SMB1 Support

SMB1 support currently cannot be disabled in YNQ Server, next build release will include this feature.

4.7.3 SMB2 Support

SMB2 support has become the default dialect for communication, can be disabled (leading to SMB3 and SMB311 being disabled), not recommended, since SMB1 only protocol can be not used by many Windows hosts due to security issues.

When YNQ is compiled with SMB2 (and above) support, the decision which SMB dialect to use is taken according to the following rules:

- If client sends SMB2 Negotiate, YNQ SMB Server starts SMB2 session. Next Negotiate exchange determines the final SMB version used according to max version supported by both server and client.
- If client sends SMB1 Negotiate, YNQ SMB Server analyses the list of supported dialects sent by client. If this list contains "SMB 2.002" or "SMB 2.???" dialect, YNQ starts SMB2 session. Otherwise, it starts SMB1 session.

4.7.4 SMB3/SMB311 Support and Encryption

SMB3 and SMB311 support is optional and it requires more resources (see [1]). During the Negotiate phase Client send the list of supported dialects and YNQ Server chooses the max dialect supported by both sides. One of the added features of

SMB3/SMB311 is encryption, YNQ Server supports both global encryption and encryption per share.

4.7.5 Domain Membership and Pass-through Authentication

Domain membership in YNQ SMB Server is essential for modern means of pass-through authentication. In this case YNQ Server uses the method of pass-through authentication that grants message signing capabilities.

4.7.6 Message Signing

4.7.6.1 Policy

YNQ SMB Server supports message signing. A client and YNQ Server mutually decide on message signing. The default message signing policy is defined by the `UD_CS_MESSAGESIGNINGPOLICY` parameter (see [1]). The function `csCtrlSetMessageSigningPolicy()` (see [3]) allows to modify this policy during runtime. There is a difference, however, between setting `UD_CS_MESSAGESIGNINGPOLICY` to zero in compile time and reducing this policy to zero after it was initially set to 1 or 2. In the first case YNQ Server is compiled with message signing support and a call to the function `csCtrlSetMessageSigningPolicy()` has no effect. In the last case, message signing mechanism is enabled yet remains inactive.

Note: The last method is the only way to accept a Windows 8 connection without signing. This is essential since Windows 8 unexpectedly sends selected SMBs with message signatures and expects a signed response from YNQ Server even both sides had agreed on no signing. When message signing is enabled but inactive, YNQ responds with a signed response even though message signing should not happen.

4.7.6.2 Message Signing and Domain Membership

This section is relevant only for YNQ Client Corporate and YNQ Server Corporate. The message signing mechanism is tightly connected with pass-through mechanism. Those two mechanisms are regulated (turned on/off) by the following compilation parameters:

```
UD_CS_INCLUDEPASSTHROUGH
UD_CS_MESSAGESIGNINGPOLICY
UD_CS_INCLUDEDOMAINMEMBERSHIP
UD_CC_INCLUDEDOMAINMEMBERSHIP
```

See [1] for more details concerning those parameters. YNQ SMB Server negotiates message signing differently according to those parameters. The following table reflects the logon results depending on parameters and other conditions.

Case	1	2	3	4	5
<code>UD_CS_INCLUDEPASSTHROUGH</code>		+	+	+	-
<code>UD_CS_MESSAGESIGNINGPOLICY</code>	0	1 or 2	1 or 2	1 or 2	1 or 2
<code>UD_CS_INCLUDEDOMAINMEMBERSHIP</code>		+	+	+	
<code>UD_CC_INCLUDEDOMAINMEMBERSHIP</code>		+	+	+	
Domain join result	Any	Success	Failure	None	Any
Logon type	Any	Domain	Local	Domain	Local
Is traffic is signed? (when client agrees)	No	Yes	Yes	No	Yes

Legend:

“+”	The parameter is defined
“-“	The parameter is not defined
<blank>	The parameter value can be any
“success”	YNQ Server joined domain and the secret, granted by DC during joining is a valid value. See <code>ccDomainJoin</code> in [3] .
“failure”	YNQ Server joined domain but the secret has invalid value. See <code>ccDomainJoin</code> in [3] .
“none”	YNQ Server did not join the domain.
“any”	Logon results do not depend on domain join results.

As can be seen from the table above, YNQ Server succeeds to authenticate user on domain in cases 2 and 3. However, it uses different pass-through mechanism in each of those cases. In case 4, since YNQ Server did not join the domain, it uses an alternate mechanism that does not grant message signing. To have message signing YNQ Server should either properly join the domain or revert to local logon only. The function `csCtrlSetMessageSigningPolicy()` allows to dynamically change message signing policy. This is a dynamic method equivalent to changing the `UD_CS_MESSAGESIGNINGPOLICY` parameter. New signing policy will be applied to the next connection and it will not affect the existing connections.

4.7.7 Handling Client Connections

YNQ SMB Server can accept a limited number of connections. The maximum number of connections is a compile-time parameter - `UD_FS_NUMSERVERSESSIONS` – see [1] .

When a number of connected clients reaches this maximum value and another client computer is connecting, YNQ closes and releases “the least active” client session. “The least active session” is one with the oldest last request received.

If `UD_CS_REFUSEONSESSIONTABLEOVERFLOW` parameter is defined, when session table is full the new connection will be refused.