

# YNQ Client Integration and Porting Guide

**YNQ 1.6.0**

Document version 0.011

# Table of Contents

<b>1</b>	<b>REFERENCED DOCUMENTS .....</b>	<b>4</b>
<b>2</b>	<b>OVERVIEW .....</b>	<b>5</b>
2.1	NQ SOFTWARE .....	5
2.2	INTEGRATION PROCESS OVERVIEW .....	6
<b>3</b>	<b>NQ SOFTWARE .....</b>	<b>7</b>
<b>4</b>	<b>ISSUES ADDRESSED DURING PORTING OR INTEGRATION PROCESS .....</b>	<b>9</b>
4.1	FILE CALLS .....	9
4.2	SOCKET CALLS .....	9
4.3	OTHER CALLS .....	9
4.4	LANGUAGES SUPPORT .....	9
<b>5</b>	<b>CORE REFERENCE.....</b>	<b>11</b>
5.1	CONSTANTS .....	11
5.1.1	<i>CIFS/SMB Error Codes .....</i>	<i>11</i>
5.1.2	<i>Event Log Constants .....</i>	<i>16</i>
5.1.3	<i>Other Constants .....</i>	<i>17</i>
5.2	DATA TYPES .....	17
5.2.1	<i>Scalar Data Types.....</i>	<i>17</i>
5.2.2	<i>NQ_IPADDRESS, NQ_IPADDRESS4 and NQ_IPADDRESS6 .....</i>	<i>19</i>
5.2.3	<i>UDFileAccessEvent .....</i>	<i>20</i>
5.2.4	<i>UDShareAccessEvent .....</i>	<i>22</i>
5.3	ACCESS TO NQ_IPADDRESS .....	23
5.3.1	<i>CM_IPADDR_VERSION.....</i>	<i>24</i>
5.3.2	<i>CM_IPADDR_SIZE.....</i>	<i>25</i>
5.3.3	<i>CM_IPADDR_GET4.....</i>	<i>26</i>
5.3.4	<i>CM_IPADDR_GET6.....</i>	<i>27</i>
5.3.5	<i>CM_IPADDR_ASSIGN4.....</i>	<i>28</i>
5.3.6	<i>CM_IPADDR_ASSIGN6.....</i>	<i>29</i>
5.3.7	<i>CM_IPADDR_EQUAL4.....</i>	<i>30</i>
5.3.8	<i>CM_IPADDR_EQUAL6.....</i>	<i>31</i>
5.3.9	<i>CM_IPADDR_EQUAL.....</i>	<i>32</i>
5.3.10	<i>cmAsciiToIp.....</i>	<i>33</i>
5.3.11	<i>cmIpToAscii.....</i>	<i>34</i>
5.3.12	<i>cmIPDump.....</i>	<i>35</i>
5.4	START/STOP FUNCTIONS .....	36
5.4.1	<i>ndStart.....</i>	<i>37</i>
5.4.2	<i>ndStop .....</i>	<i>38</i>
5.4.3	<i>ccInit .....</i>	<i>39</i>
5.4.4	<i>ccShutdown .....</i>	<i>40</i>
5.4.5	<i>nsInitGuard.....</i>	<i>41</i>
5.4.6	<i>nsExitGuard.....</i>	<i>42</i>

5.5	STRING FUNCTIONS .....	43
5.5.1	<i>cmAnsiToFs</i> .....	44
5.5.2	<i>cmFsToAnsi</i> .....	45
<b>6</b>	<b>PORTING GUIDE .....</b>	<b>46</b>
6.1	PORTING OVERVIEW .....	46
6.2	STARTUP (ST) MODULE .....	47
6.3	FILE SYSTEM (FS) MODULE .....	48
6.3.1	<i>cifsfsDrv</i> .....	49
6.3.2	<i>cifsfsDrvRemove</i> .....	50
6.4	SYSTEM (SY) MODULE.....	51
6.4.1	<i>OS Dependent Code</i> .....	51
6.4.2	<b><i>Compile Dependent Code</i></b> .....	227
6.4.3	<b><i>Hardware Dependent Code</i></b> .....	278
6.4.4	<b><i>Trace System</i></b> .....	283
<b>7</b>	<b>INTEGRATION GUIDE.....</b>	<b>294</b>
7.1	INTEGRATION PROCESS .....	294
7.2	RUN-TIME FUNCTIONS .....	294
7.2.1	<i>Synchronization Calls</i> .....	295
7.2.2	<i>Run-Time Parameters</i> .....	300
7.2.3	<i>Domain Membership</i> .....	331
7.2.4	<i>Event Log</i> .....	334
7.2.5	<i>Error Code Conversions</i> .....	337
7.2.6	<i>Code Page Conversion</i> .....	340
7.2.7	<i>Activate or deactivate SMB 1 dialect</i> .....	342
7.3	COMPILE-TIME PARAMETERS .....	344
7.3.1	<i>Functional Parameters</i> .....	344
7.3.2	<i>Performance Tuning</i> .....	401
<b>8</b>	<b>IMPLEMENTATION MANUAL.....</b>	<b>427</b>
8.1	BUFFER ALLOCATION .....	427
8.2	CHOOSING BUFFER SIZE .....	427
8.3	64-BIT FILE OFFSETS .....	428

## Table of Figures

FIGURE 1. NQ SOFTWARE COMPONENTS .....	5
FIGURE 2. NQ SOFTWARE VERTICAL DECOMPOSITION.....	7
FIGURE 3 LANGUAGE SUPPORT WITH ANSI-ONLY FILE SYSTEM .....	10
FIGURE 4 LANGUAGE SUPPORT WITH ANSI-ONLY FILE SYSTEM .....	10

## **1 Referenced Documents**

- [1] NQ User's Guide
- [2] NQ Library Reference
- [3] Common Internet File System (CIFS/SMB). Technical Reference. Revision: 1.0. SNIA
- [4] NQ Library Reference - LDAP

## 2 Overview

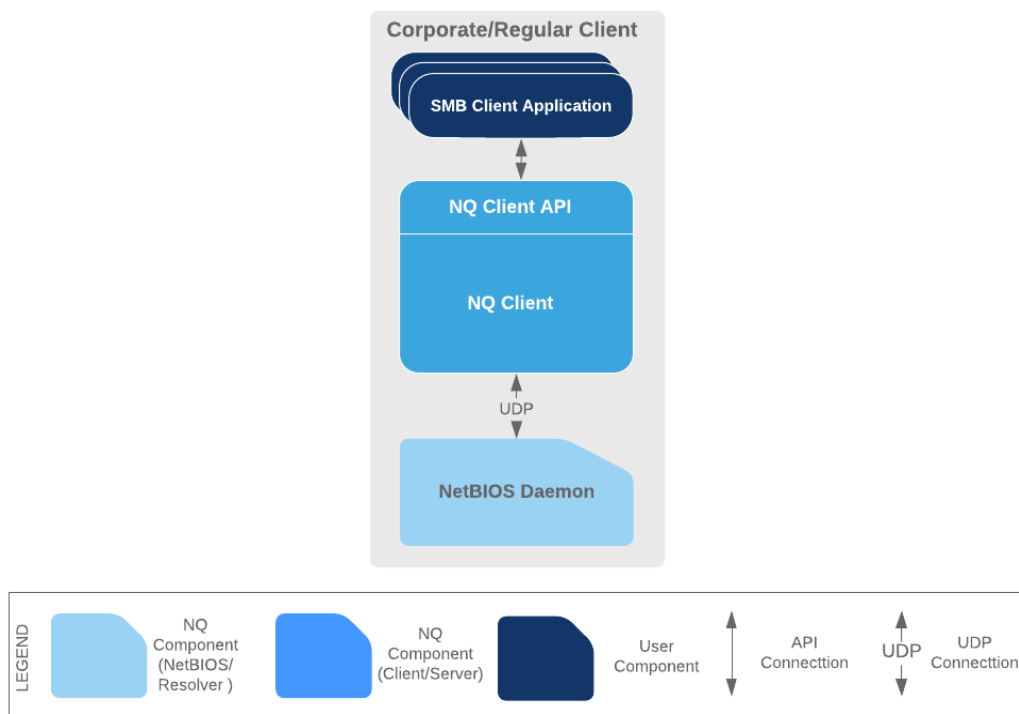
Adapting NQ software to user's project may require two processes: porting and integration. Those procedures involve several components of the entire NQ software which are intended for user's modification. A user may decide to modify implementation of some functions or values of certain parameters. Sometimes, a user may be interested to inspect implementation of other functions involved in porting and integration and view the values of relevant parameters.

Functions and parameters that a user may either modify or inspect are described in this document.

### 2.1 NQ Software

NQ package is a software component that is developed to be incorporated into software projects. Making NQ a part of a user's project is called *integration*.

NQ software includes three major components: CIFS/SMB *Server*, CIFS/SMB *Client* and *NetBIOS*. First two components are optional. A project, featuring CIFS/SMB may contain either CIFS/SMB server, or CIFS/SMB client or both. The NetBIOS component is mandatory. It provides the first two components with NetBIOS naming service and NetBIOS transport service.



**Figure 1. NQ Software components**

Application software written by user interacts with NQ by means of:

- *Server API*. Since NQ Server does not require run-time control, this interface is rarely used. A user may use it for adding or removing shares.
- *Client API*. This is the most flexible and the most comprehensive way to utilize all capabilities of NQ Client. This is also the only way to benefit from Browser and Unicode capabilities.

- *NQ File system driver.* This is the most convenient way to benefit from NQ Client. Being framed as a file system driver, NQ Client exposes a remote file system as yet another local drive. This method, however, does not give access to Browser and Unicode capabilities.

NQ software is supplied as either binary or source code. Source code software features the complete flexibility of porting from OS to OS and per-project customization. Binary code software cannot be ported and its per-project customization capability is limited.

## **2.2 Integration Process Overview**

NQ software is both portable and customizable. NQ *portability* stands for its potential to be adapted to another software/hardware platform. Here "platform" stands for a combination of 1) operating system (OS), 2) hardware, and 3) the compiler. The portability process is documented in section 4. Developers may use the result of this process for all products running on the same platform. Source code license is required for porting.

*Customization* is an adjustment of NQ software for a specific project. The process of customization comprises the following:

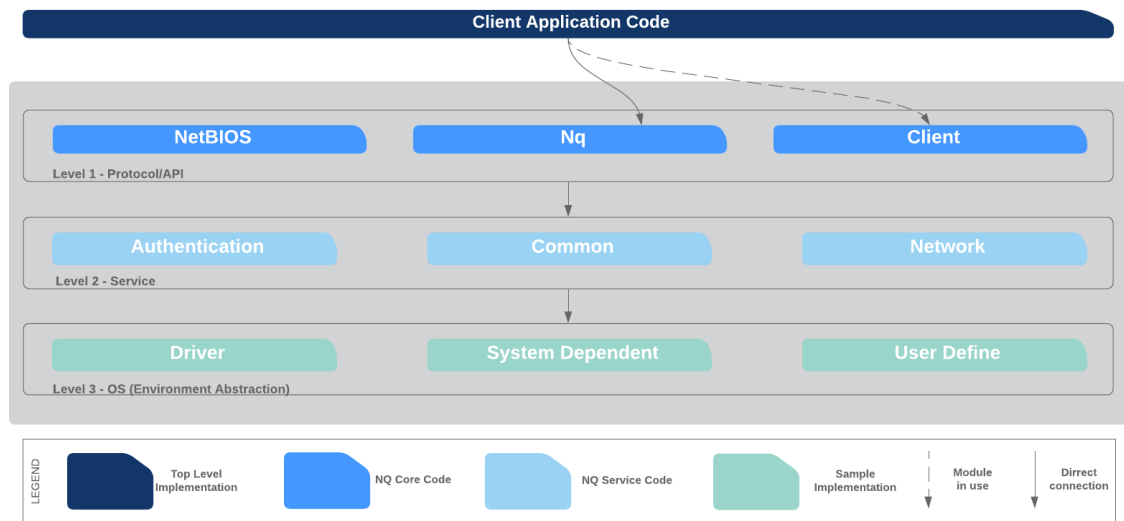
1. Optimizing NQ behavior for a specific combination of software and hardware resources. Source code license is required for optimization.
2. Providing NQ with resources whose source may be (but not necessarily is) project-dependent. Binary code license is sufficient for customizing these features.

Platform-dependent and project-dependent modules (see 3) comprise relatively small amount of code (about 5% of the entire NQ code). A user benefits from platform-independent capabilities and can adjust flexible platform-dependent features.

Though NQ features hundreds of optimization parameters, a developer does not necessarily need to redefine all of them. Usually, a developer is interested in tuning not more than 10% of the entire optimization capabilities. For this reason, NQ provides developers with the most common default values. Therefore, customizing process and porting process look much like revising code rather than changing it. The second task (providing resources) also has a default implementation in NQ software. This implementation is suitable for a typical embedded project.

### 3 NQ Software

Entire NQ software is divided into several modules. A module is designated by two-letter mnemonics. Most of NQ modules are both platform-independent and project-independent that takes them out of this document's scope. Four modules belong, however, to this document's framework.



**Figure 2. NQ Software Vertical Decomposition**

This figure demonstrates a vertical layout of NQ, where:

Core layer contains multiple platform-independent and project-independent modules of NQ

Startup (ST) module runs NQ tasks. This component is platform-dependent and it may be project-dependent also.

Filesystem (FS) module creates a platform-dependent driver framework for NQ Client.

System (SY) module implements platform-dependent library.

Project (UD) module is project-dependent library

The sources of NQ software are entirely written in C language. The layout above reflects code dependencies:

- UD module does not depend on other modules,
- SY depends on UD only,
- Core modules depend on SY and UD,
- ST depends on core modules.
- FS depends on core modules and on SY

NQ software uses "system layer abstraction" concept to make its software platform-independent. Core modules issue calls that are platform-independent, while SY and UD implement those calls in a platform-dependent way. A "call" may be either a macro call or a function call. Macro call is less time-consuming however some calls cannot be implemented this way. Many of platform-dependent calls use BSD-like notation and may be easily mapped on appropriate BSD calls.

NQ software has internal resources: buffers, queues, etc. The initial amount of those resources is specified in the UD module. The same module provides NQ with external

resources, such as host IP, network mask, shares, etc. The way, those resources are defined may vary from project to project.



## 4 Issues Addressed During Porting or Integration Process

This paragraph designates the process of NQ porting and/or integration into smaller themes. Since the scope of this paragraph is rather sketchy, it allies mainly to the porting process, while few aspects here apply to the integration process.

### 4.1 File Calls

NQ applies to the underlying operating system for file operations: creating, deleting, reading, writing, positioning, etc. The process of porting NQ to yet another operating system involves delegating the calls to NQ system abstraction level to file system calls.

Most of file calls in NQ have syntax similar to that of POSIX file calls that allows translating them one-to-one into appropriate POSIX calls. The only exception is directory scanning which may be implemented by two POSIX calls.

### 4.2 Socket Calls

NQ assumes that the underlying operating system has socket interface or its equivalent.

As with file calls, socket calls of the NQ system abstraction level correspond to BSD socket calls.

### 4.3 Other Calls

NQ uses very few calls that do not fall in the two above categories.

### 4.4 Languages Support

This paragraph addresses NQ capability of supporting file names in different languages. Multiple languages can be supported in two ways:

1. Codepage support where names are converted to/from national ANSI extension. Codepage stands here for transition between Unicode<sup>1</sup> and national ANSI and vice versa. This method supports one language (codepage) for an installation.
2. Unicode support. This method allows supporting multiple languages at the same time.

The two methods above can be combined. Choosing proper method(s) involves the following factors:

- NQ software is capable of keeping all names internally in Unicode. However, the other side of a CIFS/SMB connection should also support Unicode. If the second side of the connection send information in ANSI or/and does not accept Unicode names, then the only possibility is codepage support<sup>2</sup>
- An ANSI-only file system may have restrictions on the entire set of 255 ANSI characters because some of them it uses for internal purposes. Usually, those restricted characters are so-called “non-printable” symbols that are not included in most codepages. Some Asian language, however, may use those symbols as the second bytes of two-byte characters. NQ software grants a

---

<sup>1</sup> UTF-16 Little Endian

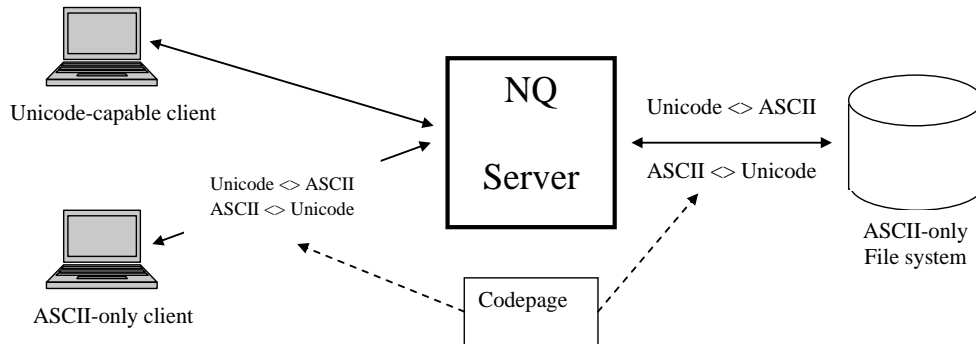
<sup>2</sup> Windows 95 always sends information in ASCII.

mechanism of avoiding this problem, yet some integration efforts are expected here.

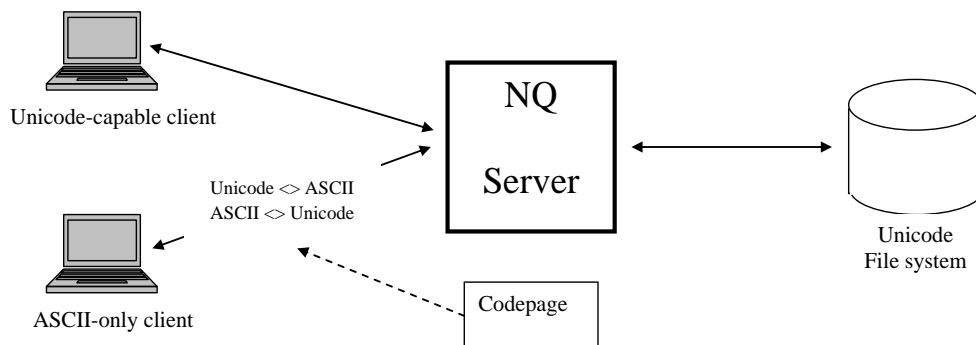
In the combined case, NQ software will support multiple languages for a Unicode-capable connection and will supply one language only for an ANSI connection.

Either of two methods above requires NQ software to be generated with Unicode support.

Next two pictures illustrate language support with ANSI-only and Unicode-capable file systems respectively.



**Figure 3 Language support with ANSI-only file system**



**Figure 4 Language support with Unicode file system**

Language support schema for NQ Client is similar except for the following differences:

1. No file system is involved.
2. NQ exposes so-called triplets for those API calls that require string parameters. See [2] and [1] for details. Calling a triplet function may cause ANSI to/from Unicode conversion. This is, however, out of this document's scope.
3. When NQ Client is extended with file system driver (see 6.3), more restrictions might apply since the driver interface in the target operating system might not support Unicode.

## 5 Core Reference

This paragraph designates those NQ functions, constants and data types that are referenced or may be used in the porting or integration process. They belong to the Core NQ and, as those, should not be modified

### 5.1 Constants

#### 5.1.1 CIFS/SMB Error Codes

This paragraph contains definitions for error code that a CIFS/SMB server may return to a CIFS/SMB client. Most error codes in this paragraph designate so-called “DOS” error codes, while others belong to “NT” error codes. Integration/porting process may involve using these codes when implementing error code conversion. See 7.2.5 for more details. Error code constants are defined in the *udapi.h* file.

DOS error codes:

Name	Comment
SMB_ERRsuccess	No error
DOS_ERRbadfunc	Invalid function. The server did not recognize or could not perform a system call generated by the server, e.g., set the DIRECTORY attribute on a data file, invalid seek mode.
DOS_ERRbadfile	File not found. The last component of a file's pathname could not be found.
DOS_ERRbadpath	Directory invalid. A directory component in a pathname could not be found.
DOS_ERRnofids	Too many open files
DOS_ERRnoaccess	Access denied, the client's context does not permit the requested function. This includes the following conditions: invalid rename command, write to Fid open for read only, read on Fid open for write only, attempt to delete a non-empty directory.
DOS_ERRbadfid	Invalid file handle. The file handle specified was not recognized by the server.
DOS_ERRbadmcb	Memory control blocks destroyed
DOS_ERRnomem	Insufficient server memory to perform the requested function.
DOS_ERRbadmem	Invalid memory block address.

DOS_ERRbadenv	Invalid environment.
DOS_ERRbadformat	Invalid format.
DOS_ERRbadaccess	Invalid open mode.
DOS_ERRbaddata	Invalid data (generated only by IOCTL calls within the server).
DOS_ERRbaddrive	Invalid drive specified
DOS_ERRremcd	A Delete Directory request attempted to remove the server's current directory.
DOS_ERRdiffdevice	Not same device (e.g., a cross volume rename was attempted).
DOS_ERRnofiles	A File Search command can find no more files matching the specified criteria.
DOS_ERRbadshare	The sharing mode specified for an Open conflicts with existing FIDs on the file.
DOS_ERRlock	A Lock request conflicted with an existing lock or specified an invalid mode, or an Unlock requested attempted to remove a lock held by another process.
DOS_ERRdontsupportipc	Server does not implement RPC calls.
DOS_ERRnoshare	Requested share does not exist on the server
DOS_ERRfileexists	File exists while the required operation (e.g., - rename) expects it not.
DOS_ERRbaddirectory	
DOS_ERRinsufficientbuffer	Server is out of resources
DOS_ERRinvalidname	File or path name contains illegal characters
DOS_ERRdirnotempty	Server was unable to delete a non-empty directory
DOS_ERRalreadyexists	New file cannot be created since a file with the same name already exists and cannot be overwritten.
DOS_ERRbadpipe	Error on names pipe.
DOS_ERRpipebusy	Pipe cannot perform the required operation due to a lack of resources.
DOS_ERRpipeclosing	Pipe is being closed and cannot perform the required operation.
DOS_ERRnotconnected	Some resource on server was not connected for the required operation.

DOS_ERRmoredata	Some data in the response did not fit into the buffer provided by client.
SRV_ERRerror	Non-specific error code. It is returned under the following conditions: resource other than disk space exhausted (e.g. TIDs), first SMB command was not negotiated, multiple negotiates attempted, and internal server error.
SRV_ERRbadpw	Bad password - name/password pair in a Tree Connect or Session Setup are invalid.
SRV_ERRaccess	The client does not have the necessary access rights within the specified context for the requested function.
SRV_ERRinvtid	The Tid specified in a command was invalid.
SRV_ERRinvnetname	Invalid network name in tree connect.
SRV_ERRinvdevice	Invalid device - printer request made to non-printer connection or non-printer request made to printer connection.
SRV_ERRqfull	Print queue full (files) -- returned by Open Print File.
SRV_ERRqtoobig	Print queue full -- no space.
SRV_ERRqeof	EOF on print queue dump.
SRV_ERRinvfid	Invalid print file FID.
SRV_ERRsmbcmd	The server did not recognize the command received.
SRV_ERRsrverror	The server encountered an internal error, e.g., system file unavailable.
SRV_ERRfilespecs	The FID and pathname parameters contained an invalid combination of values.
SRV_ERRbadpermits	The access permissions specified for a file or directory is not a valid combination. The server cannot set the requested attribute.
SRV_ERRsetattrmode	The attribute mode in the Set File Attribute request is invalid.
SRV_ERRpaused	Server is paused. (Reserved for

	messaging).
SRV_ERRmsgoff	Not receiving messages. (Reserved for messaging).
SRV_ERRnoroom	No room to buffer message (Reserved for messaging).
SRV_ERRrmuns	Too many remote user names (Reserved for messaging).
SRV_ERRtimeout	Operation timed out.
SRV_ERRnoresource	No resources currently available for request.
SRV_ERRtoomanyuids	Too many UID's active on this session.
SRV_ERRinvuid	The UID is not known as a valid user identifier on this session.
SRV_ERRusempx	Temporarily unable to support Raw, use MPX mode.
SRV_ERRusestd	Temporarily unable to support Raw, use standard read/write.
SRV_ERRcontmpx	Continue in MPX mode.
SRV_ERRnosupport	Function not supported.
HRD_ERRnowrite	Attempt to write on write- protected media
HRD_ERRbadunit	Unknown unit.
HRD_ERRnotready	Drive not ready.
HRD_ERRbadcmd	Unknown command.
HRD_ERRdata	Data error (CRC).
HRD_ERRbadreq	Bad request structure length.
HRD_ERRseek	Seek error.
HRD_ERRbadmedia	Unknown media type.
HRD_ERRbadsector	Sector not found.
HRD_ERRnopaper	Printer out of paper.
HRD_ERRwrite	Write fault.
HRD_ERRread	Read fault.
HRD_ERRgeneral	General failure.
HRD_ERRbadshare	An open conflict with an existing open.
HRD_ERRlock	A Lock request conflicted with an existing lock or specified an invalid mode, or an Unlock requested attempted to remove a lock held by another process.
HRD_ERRwrongdisk	The wrong disk was found in a drive.
HRD_ERRFCBUnavail	No FCB's are available to process request.
HRD_ERRsharebufexc	A sharing buffer has been exceeded.
HRD_ERRdiskfull	Media is full.

NT status codes:

<b>Name</b>	<b>Comment</b>
SMB_STATUS_OK	No error.
SMB_STATUS_CANCELLED	Notify request was cancelled by the server by client request or internally.
SMB_STATUS_ACCESS_DENIED	Server denied access due to insufficient access right or wrong open mode.
SMB_STATUS_OBJECT_NAME_NOT_FOUND	Server failed to provide the requested data.
SMB_STATUS_INVALID_PARAM	Request contains illegal parameter.
SMB_STATUS_LOGON_FAILURE	Server was unable to logon client. Probably wrong user name or password.
SMB_STATUS_BAD_NETWORK_PATH	Path error in a client request.
SMB_STATUS_BAD_NETWORK_NAME	Client requested for an illegal or non-existing share.
SMB_STATUS_MORE_ENTRIES	Some data did not fit into the server response.
SMB_STATUS_ENUMDIR	This is a status rather than error code. The Notify response is Enumerate Directory.
SMB_STATUS_BUFFER_OVERFLOW	RPC buffer overflow.

### 5.1.2 Event Log Constants

Constants designated in this paragraph are used in event logging (see 7.2.2.22). Event log constants are defined in the *udapi.h* file.

Name	Value	Comment
UD_LOG_MODULE_CC	2	Designates NQ Client as the event origin
UD_LOG_CLASS_GEN	1	Event class for start and stop events
UD_LOG_CLASS_FILE	2	Event class for file/directory access events
UD_LOG_CLASS_SHARE	3	Event class for share connect/disconnect events
UD_LOG_GEN_START	1	Start event type
UD_LOG_GEN_STOP	2	Stop event type
UD_LOG_FILE_CREATE	1	File/directory create event type
UD_LOG_FILE_OPEN	2	File/directory open event type
UD_LOG_FILE_CLOSE	3	File/directory close event type
UD_LOG_FILE_DELETE	4	File/directory delete event type
UD_LOG_FILE_RENAME	5	File/directory rename event type
UD_LOG_FILE_ATTRIB	6	File/directory change attributes event type
UD_LOG_SHARE_CONNECT	1	Share connection event type
UD_LOG_SHARE_DISCONNECT	2	Share disconnection event type



### 5.1.3 Other Constants

This paragraph contains those constants that do not fit in other categories. These constants are defined in the *cmcommon.h* file.

Name	Value	Comment
CM_IPADDR_IPV4	4	Designates IPv4 address
CM_IPADDR_IPV6	6	Designates IPv6 address
NQ_SUCCESS	0	Call succeeded
NQ_FAIL	-1	Call failed (general failure)
TRUE	1	If not defined by the compiler
FALSE	0	If not defined by the compiler
NULL	0	If not defined by the compiler
UD_DNS_SERVERSTRINGSIZE	See below	Maximum size of the DNS server string (see 7.2.2.4).

NQ\_SUCCESS and NQ\_FAIL are used as generic return codes. A non-zero error code may be used instead of NQ\_FAIL. See also 5.2.1 .

The macro UD\_DNS\_SERVERSTRINGSIZE is calculated in compile time as a multiplier of maximum length of IP address in NQ\_WCHAR characters, number of adapter and number of DNS servers. This should include enough space for semicolon symbols and a terminating zero.

## 5.2 Data Types

This section designates data types referenced during Integration/Porting and functions that may be used during that process.

### 5.2.1 Scalar Data Types

The data types below are defined in the *udapi.h* file.

Name	Description
NQ_CHAR	One byte ACSII character
NQ_BYTE	Abstract data byte
NQ_INT	Generic-purpose integer
NQ_UINT	Generic-purpose unsigned integer
NQ_BOOL	TRUE or FALSE
NQ_INDEX	Integer used as a cycle variable
NQ_COUNT	Return value for functions returning a data count
NQ_INT16	Signed 16 bit integer value
NQ_UINT16	Unsigned 32 bit integer value
NQ_INT32	Signed 32 bit integer value
NQ_UINT32	Unsigned 32 bit integer value
NQ_WCHAR	“Wide-character” used for Unicode
NQ_HANDLE	Pointer to a context-abstract structure
NQ_STATUS	NQ_SUCCES or NQ_FAIL
NQ_TIME	32-bit Unix time
NQ_PORT	Port number in Network Byte Order
NQ_IPADDRESS4	IPv4 address – loadable value

NQ_IPADDRESS6	IPv6 address – an array
---------------	-------------------------

### 5.2.2 NQ\_IPADDRESS, NQ\_IPADDRESS4 and NQ\_IPADDRESS6

This type designates an abstract IP address that may be either IPv4 or IPV6. The actual definition of this type depends on compile-time parameters (see 7.3.1.1, 7.3.1.1 and 7.3.1.4) and, therefore, it cannot be specified here. This type is not loadable and should not be used directly but rather through access calls (see 5.3).

The concrete IP addresses are defined by the following types found in *udapi.h*:

Name	Defines as
NQ_IPADDRESS4	NQ_UINT32
NQ_IPADDRESS6	NQ_UINT16[8]

Both types above contain values in the Network Byte Order. It is also important to emphasize that IPv4 address is a loadable value in contrary to IPv6 address.

### 5.2.3 UDFileAccessEvent

This structure is defined in the *udapi.h* file.

#### Prototype

```
typedef struct {  
    const NQ_WCHAR *fileName;  
    const NQ_WCHAR *newName;  
    NQ_UINT32 access;  
} UDFileAccessEvent;
```

#### Description

This structure keeps data for an event from File/Directory Access class. Depending on the event type, some members of this structure may be undefined.

#### Members

filename

Pointer to the file name. For UD\_LOG\_FILE\_RENAME event this value designates old file name. This value may be NULL when an error was encountered in converting the file name to the host form.

newName

Pointer to the new file name. This value is used only with the UD\_LOG\_FILE\_RENAME event and it designates file name after rename operation. This value may be NULL when an error was encountered in converting this name to the host form.

access

This value has different meaning for different events:

- For UD\_LOG\_FILE\_CREATE, UD\_LOG\_FILE\_DELETE and UD\_LOG\_FILE\_RENAME – no meaning.
- For UD\_LOG\_FILE\_OPEN and UD\_LOG\_FILE\_CLOSE this value means file open rights as follows:
  - 0x0 – read
  - 0x1 – write
  - 0x2 – read/write
  - 0x3 – execute
  - 0x8000 - delete
  - 0xF – allOnly four least significant bits are important. Other bits may contain any value.
- For UD\_LOG\_FILE\_ATTRIB this value means new file attributes as a combination of the following bits:
  - 0x1 – readonly
  - 0x2 – hidden
  - 0x4 – system

0x8 – volume  
0x10 – directory  
0x20 - archive  
0x80 – normal

A value of -1 in this parameter may occur on error only and it means that the error condition was encountered regardless of file attributes.

#### 5.2.4 UDShareAccessEvent

This structure is defined in the *udapi.h* file.

##### Prototype

```
typedef struct {  
    const NQ_WCHAR *shareName;  
    NQ_BOOL ipc;  
    NQ_BOOL printQueue;  
} UDShareAccessEvent;
```

##### Description

This structure keeps data for an event from Share Access class. Depending on the event type, some members of this structure may be undefined.

##### Members

shareName

Pointer to the share name.

ipc

This value is TRUE for IPC\$ share and FALSE for a share of another type

printQueue

This value is TRUE for a printer queue share and FALSE for a share of another type

### **5.3 Access to NQ\_IPADDRESS**

The NQ\_IPADDRESS structure (see 5.2.2) is defined differently depending on compilation parameters. NQ Core modules provide integrator with common means to access the following components of this structure:

1. IPv4 address
2. IPv6 address

Since some of the calls below are macros they should not be used inside expressions. The calls of this category are defined in the *cmcommon.h* file.

### 5.3.1 CM\_IPADDR\_VERSION

#### Prototype

```
NQ_INT CM_IPADDR_VERSION(  
    const NQ_IPADDRESS addr  
);
```

#### Description

Get IP address type

#### Parameters

addr	IN
IP address	

#### Return Value

CM\_IPADDR\_IPV4 or CM\_IPADDR\_IPV6

#### Notes

This call allows determining the concrete type of an abstract IP address.



### 5.3.2 CM\_IPADDR\_SIZE

#### Prototype

```
NQ_INT CM_IPADDR_SIZE(  
    const NQ_IPADDRESS addr  
);
```

#### Description

Get the actual size of an IP address

#### Parameters

addr	IN
IP address	

#### Return Value

Actual size of the address

#### Notes

This call returns the number of bytes needed to hold the actual address.

### 5.3.3 CM\_IPADDR\_GET4

#### Prototype

```
NQ_IPADDRESS4 CM_IPADDR_GET4 (  
    const NQ_IPADDRESS addr  
);
```

#### Description

Withdraw an IPv4 address from an abstract IP address

#### Parameters

addr	IN
	IP address

#### Return Value

IPv4 address

#### Notes

An IPv4 address is a loadable value

#### 5.3.4 CM\_IPADDR\_GET6

##### Prototype

```
NQ_IPADDRESS6 CM_IPADDR_GET6(  
    const NQ_IPADDRESS addr  
);
```

##### Description

Withdraw an IPv6 address from an abstract IP address

##### Parameters

addr	IN
IP address	

##### Return Value

Pointer to IPv6 address

##### Notes

An IPv6 address is a pointer

### 5.3.5 CM\_IPADDR\_ASSIGN4

#### Prototype

```
void CM_IPADDR_ASSIGN4(  
    NQ_IPADDRESS addr,  
    NQ_IPADDRESS4 ip4  
);
```

#### Description

Set IP address as IPv4 address

#### Parameters

addr	OUT
	IP address
ip4	IN
	IPv4 address to assign to the abstract IP address

#### Return Value

None

#### Notes

### 5.3.6 CM\_IPADDR\_ASSIGN6

#### Prototype

```
void CM_IPADDR_ASSIGN6(  
    NQ_IPADDRESS addr,  
    const NQ_IPADDRESS6 ip6  
);
```

#### Description

Set IP address as IPv6 address

#### Parameters

addr	OUT
	IP address
ip6	IN
	IPv6 address to assign to the abstract IP address

#### Return Value

None

#### Notes

### 5.3.7 CM\_IPADDR\_EQUAL4

#### Prototype

```
NQ_BOOL CM_IPADDR_EQUAL4 (  
    NQ_IPADDRESS addr,  
    NQ_IPADDRESS4 ip4  
);
```

#### Description

Compare an abstract IP address with IPv4 address

#### Parameters

addr	IN
IP address	
ip4	IN
IPv4 to compare	

#### Return Value

TRUE when abstract IP address is the same as IPv4 address, FALSE otherwise

#### Notes

Abstract address should be of IPv4 type and its address value should match the IPv4 address

### 5.3.8 CM\_IPADDR\_EQUAL6

#### Prototype

```
NQ_BOOL CM_IPADDR_EQUAL6 (  
    NQ_IPADDRESS addr,  
    NQ_IPADDRESS6 ip6  
);
```

#### Description

Compare an abstract IP address with an IPv6 address

#### Parameters

addr	IN
IP address	
ip6	IN
IPv6 address to assign to the abstract IP address	

#### Return Value

TRUE when abstract IP address is the same as IPv6 address, FALSE otherwise

#### Notes

Abstract address should be of IPv6 type and its address value should match the IPv6 address

### 5.3.9 CM\_IPADDR\_EQUAL

#### Prototype

```
NQ_BOOL CM_IPADDR_EQUAL(  
    NQ_IPADDRESS addr,  
    NQ_IPADDRESS other  
);
```

#### Description

Compare two abstract IP addresses

#### Parameters

addr	IN
First IP address	
other	IN
Second IP address	

#### Return Value

TRUE when both addresses are equal, FALSE otherwise

#### Notes

Abstract addresses are equal when they are of the same type and their address values are equal.



### 5.3.10 cmAsciiToIp

#### Prototype

```
NQ_STATUS cmAsciiToIp(  
    NQ_CHAR *str  
    NQ_IPADDRESS *addr  
);
```

#### Description

Convert string representation of IP address into an abstract structure

#### Parameters

str	IN
	String containing text representation of IP address
addr	OUT
	IP address

#### Return Value

NQ\_SUCCESS when conversion was done, NQ\_FAIL otherwise

#### Notes

This function converts IP address according to its type

### 5.3.11 cmIpToAscii

#### Prototype

```
NQ_STATUS cmIpToAscii(  
    NQ_CHAR *buffer  
    const NQ_IPADDRESS *addr  
);
```

#### Description

Convert an abstract IP address into printable text representation

#### Parameters

buffer	OUT
	Text buffer of enough size
addr	IN
	IP address

#### Return Value

NQ\_SUCCESS when conversion was done, NQ\_FAIL otherwise

#### Notes

This function converts IP address according to its type

### 5.3.12 cmIPDump

#### Prototype

```
NQ_CHAR *cmIPDump (  
    const NQ_IPADDRESS *addr  
);
```

#### Description

Converts IP address to a text representation into an embedded buffer

#### Parameters

addr	IN
IP address	

#### Return Value

Pointer to a text representation of the IP address

#### Notes

This function converts IP address according to its type. Since the result resides in a static buffer this function should not be used twice as a parameter in a *printf* call.

#### **5.4    *Start/Stop Functions***

Start function stands for an entry point into an NQ task. Stop function is a call to stop an NQ task.

### 5.4.1 ndStart

#### Prototype

```
NQ_STATUS ndStart(  
    void  
);
```

#### Description

Start NQ NetBIOS Daemon

#### Parameters

None

#### Return Value

NQ\_SUCCESS on success, NQ\_FAIL on error

#### Notes

A call to this function starts NQ NetBIOS Daemon. After NQ NetBIOS Daemon is fully operational it calls *udNetBiosDaemonStarted()* (see 7.2.1.1). During this period no other calls to *ndStart()* or *ndStop()* can be issued. This function returns after NQ NetBIOS Daemon is fully started.

### 5.4.2 ndStop

#### Prototype

```
void ndStop(  
    void  
);
```

#### Description

Shutdown NQ NetBIOS Daemon

#### Parameters

None

#### Return Value

None

#### Notes

After this function returns control, NQ NetBIOS Daemon task may be still active. It requires several seconds to close all sockets and perform necessarily clean-up. During this period no other calls to `ndStart()` or `ndStop()` can be issued. After this is done NQ NetBIOS Daemon calls `udNetBiosDaemonClosed()` (see 7.2.1.2).

### 5.4.3 cclnit

#### Prototype

```
NQ_BOOL ccInit(  
    void (*fsdNotify)(NQ_INT eventId, NQ_UINT32  
        param)  
    );
```

#### Description

Initialize NQ Client Library

#### Parameters

<code>fsdNotify</code>	IN
Pointer to a callback function, see Notes below	

#### Return Value

TRUE on success, FALSE on error

#### Notes

This function initializes library resources and installs a handle notification function. Handle notification is useful when NQ handle (either a file handle or a search handle) is "wrapped" into a system-dependent handle. This technique is used in file-system drivers (see 6.3). Then, when a handle is unexpectedly disposed by NQ Client it should be accordingly disposed inside a driver. NQ Client notifies driver of a disposed handle by means of callback function, specified as a parameter to *ccInit()* call. The first parameter of this function may be as follows:

- NOTIFY\_SRCH\_HANDLE\_ERROR – the notified handle is a search handle
- NOTIFY\_FILE\_HANDLE\_ERROR – the notified handle is a file handle.

The second parameter is a notified handle itself.

If no handle notification is required, this parameter may be set to NULL.

#### 5.4.4 ccShutdown

##### Prototype

```
void ccShutdown(  
    void  
);
```

##### Description

Close NQ Client Library

##### Parameters

None

##### Return Value

None

##### Notes



#### 5.4.5 nsInitGuard

##### Prototype

```
void nsInitGuard(  
    void  
);
```

##### Description

Static initialization of NetBIOS library

##### Parameters

None

##### Return Value

None

##### Notes

This call should be issued prior to any other call, except for *udInit()* (see 7.2.2.1).

#### 5.4.6 nsExitGuard

Prototype

```
void nsExitGuard(  
    void  
);
```

Description

Close NetBIOS library

Parameters

None

Return Value

None

Notes

This call should be issued after all NetBIOS clients (CIFS/SMB Client and Browser Daemon) have been closed.

## **5.5 String Functions**

This category includes functions for manipulation with Unicode (UTF-16LE) strings, ANSI strings and Unicode (UTF16LE) strings (see [1] ) as well as function for conversion between strings of different type.

NQ string functions are aware of national ANSI encoding, including variable-length national characters. For this reason NQ software supports code page definition (see 4.4 and 7.2.6) As an additional feature, NQ string functions support “file-system-dependent” encoding (see 5.5.1 and 5.5.2)

Since some of string functions are implemented as macros it is not recommended to use them in complex expressions.

### 5.5.1 cmAnsiToFs

#### Prototype

```
void cmAnsiToFs(  
    NQ_CHAR *str  
    NQ_INT size  
);
```

#### Description

Replaces “illegal” characters in an ANSI string

#### Parameters

<code>str</code>	IN/OUT
	String for in-place conversion
<code>size</code>	IN
	String size in bytes

#### Return Value

None

#### Notes

Particular file systems may have restrictions on the entire set of 255 ANSI characters. While appropriate for most languages this may cause problems for Asian languages with two-byte symbol sequences.

To avoid this, NQ allows converting “problematic” codes into those that do not have meaning in the code page yet are legal for the target file system. This function implements conversion back from ANSI to file system encoding.

Two sets of illegal codes are defined in 6.4.1.9.4 and 6.4.1.9.5. A call to this function should be placed in each SY call that delegates file name to the underlying file system.

### 5.5.2 cmFsToAnsi

#### Prototype

```
void cmFsToAnsi(  
    NQ_CHAR *str  
    NQ_INT size  
);
```

#### Description

Restores “illegal” characters in an ANSI string

#### Parameters

str	IN/OUT
	String for in-place conversion
size	IN
	String size in bytes

#### Return Value

None

#### Notes

Particular file systems may have restrictions on the entire set of 255 ANSI characters. While appropriate for most languages this may cause problems for Asian languages with two-byte symbol sequences.

To avoid this, NQ allows converting “problematic” codes into those that do not have meaning in the code page yet are legal for the target file system. This function implements conversion back from file system encoding.

Before such name is returned from a file system call, NQ restores illegal characters that were replaced during file creation. Two sets of illegal codes are defined in 6.4.1.9.4 and 6.4.1.9.5. A call to this function should be placed in each SY call that receives a file name from the underlying file system.

## 6 Porting Guide

### 6.1 Porting Overview

The goal of the porting process is to suit NQ software for another combination of OS, hardware and/or C compiler (called jointly – a *platform*). This adjustment concerns the following modules: ST, FS, SY and UD. The porting process comprises of the following steps:

1. Choosing proper method of language support if required.
2. Rewriting the ST module. Usually this very small module is completely rewritten during porting.
3. Rewriting the FS module. This step is essential only when NQ Client is included in the project.
4. Revising and modifying the SY module.
5. Revising the UD module and, possibly, rewriting some of its code.
6. Recompiling and building NQ project

## 6.2 Startup (ST) Module

This module contains two files:

STDRVMAIN.C and STNDMAIN.C

These files contain the functions to start the processes, described in [2] . NQ Driver is started after running the bin file (in VxWorks after calling the *nqdrvStart()* function) and the software initialization process is finished (this may take as much as 15 seconds). NQ NetBios is started after running the bin file (in VxWorks after calling the *nqndStart()* function) and the software initialization process is finished.

Porting this module means converting task calls into their analogs in the target OS. The ST module should run the following tasks:

- NetBIOS daemon (nbd). This step should be omitted when a foreign NetBIOS is used.
- NQ Browser Daemon.

This module should also provide code for shutting down those tasks. The method of startup/shutdown depends on the target operating system. For operating systems with common address space a developer may use "spawn" call to start up necessary tasks and a "kill" method for shutting them down. If target operating system supports separate address spaces, a developer may consider creating required tasks as separate executables.

When implementing this module, a programmer has to call a number of start/shutdown functions. These functions are described in paragraph 5.

### 6.3 File System (FS) module

FS module implements a system-dependent driver framework above NQ Client. This module includes the following files:

FSDRIVER.H - driver interface

FSDRIVER.C - driver implementation

VxWorks:

The default implementation in FSDRIVER.C registers the driver in VxWorks and provides it with several functions. Those functions implement file control primitives by delegating the actual work to appropriate NQ Client API calls.

When porting this module to another OS, user's aim is to substitute VxWorks driver with the target OS' driver.

Linux:

The default implementation is based on open source FUSE library. This library must be installed on target machine.

Use the following command to compile NQ driver based on FUSE (use `-lfuse`):

**make -C src nqdriverall**

Create a local folder to be used as mount point for remote share, for example **/home/localMount**.

Run to mount:

**./nqdrv /home/localMount \\server\remoteDir**

You will be prompted to provide user credentials.

Note that NetBIOS daemon might be required for driver operation when mounting server by name.

For help on various parameters run **./nqdrv**

Successful mount operation can be observed in output of Linux command **mount**.

Now you can work on mounted folder **/home/localMount**, all file commands will be silently translated into SMB commands against remote share.

In order to stop using driver run Linux command **umount /home/localMount**



### 6.3.1 cifsfsDrv

#### Prototype

```
int cifsfsDrv(  
    const char *name  
);
```

#### Description

This function registers NQ Client driver in the target OS

#### Parameters

name	IN
Driver name	

#### Return Value

0 on success, -1 on error

#### Notes

Driver name is that root name under which reside mounts.

### 6.3.2 cifsfsDrvRemove

#### Prototype

```
int cifsfsDrvRemove(  
    void  
);
```

#### Description

This function removes the NQ Client driver

#### Parameters

None

#### Return Value

0 on success, -1 on error

#### Notes

## 6.4 System (SY) Module

Most files of SY module reside in the directory:

*<PROJECT ROOT>/src/service/os/<TARGET OS>/sy*, for instance:

*C:\Tornado\windlink\visuality\nq\target\src\service\os\vxworks\sy*

System-independent sources of SY reside in the same location.

SY consists of the following files:

SYAPI.H	This file resides in directory <i>&lt;PROJECT ROOT&gt;/src/nq</i> . It is the API definition of the module and the only file that is referenced from the core sources.
SYCOMMON.H	This file contains definitions of those structures that are used for passing data between core sources and SY.
SYINCLUD.H	Contains <i>include</i> directives required for prototyping API functions. These directives are required for translating core sources since some of SY calls are implemented as macro substitutions.
SYTRACE.H	Declares tracing macros, referenced from core sources. Normally, core sources are translated without tracing. For these reasons this file contains actual trace macros as well as dummy trace macros. The choice between the two sets of macros is controlled by compile-time parameter <i>DEBUG</i> . If this parameter is defined – actual macros are precompiled. Otherwise, dummy macros are precompiled. A user is not expected to change this file.
SYOPSYST.H	contains OS-dependent definitions.
SYCOMPIL.H	contains compile-dependent definitions.
SYPLTFRM.H	contains platform-dependent definitions.
SYPRINTR.H	contains printer related definitions.
SYPACKON.H	contains definitions turning on the structure alignment.
SYPACKOF.H	contains definitions turning off the structure alignment.
SYSASL.H	contains GSASL interface definitions.
SYLDAP.H	contains LDAP definitions.
SYOPSYST.C	contains OS-dependent implementations.
SYCOMPIL.C	contains compile-dependent implementations.
SYPLTFRM.C	contains platform-dependent implementations.
SYPRINTR.C	contains printer related implementations.
SYSASL.C	contains GSASL interface implementations.
SYLDAP.C	contains LDAP implementations

The aim of porting this module is to implement all calls below in the target OS.

### 6.4.1 OS Dependent Code

File *SYOPSYST.H* defines parameters, function prototypes, and macro substitutions whose implementation may vary between different operating systems. File *SYOPSYST.C* implements functions.

### 6.4.1.1 System Name

6.4.1.1.1 SY\_OSNAME

Prototype

SY\_OSNAME

Description

Readable name of the Operating System

Notes

#### **6.4.1.2 Framework calls**

This group contains functions called on NQ start and NQ stop. The SY level can use these two functions for allocating/release of SY-local resources.

#### 6.4.1.2.1    syInit

##### Prototype

```
NQ_BOOL syInit(  
    void  
);
```

##### Description

Initialize SY module

##### Parameters

None

##### Return Value

TRUE on success, FALSE on error

##### Notes

SY module may use this function for initializing resources, e.g., - allocating bug buffers. This function is not referenced from NQ core. It is expected to be called from the NQ framework (see a sample code for STMAIN.H). Therefore, implantation of this function is optional and depends on SY implementation.

## 6.4.1.2.2 syStop

## Prototype

```
void syStop(  
    void  
);
```

## Description

Stop SY module

## Parameters

None

## Return Value

None

## Notes

SY module may use this function for releasing resources, e.g., - freeing bug buffers. This function is not referenced from NQ core. It is expected to be called from the NQ framework (see a sample code for STMAIN.H). Therefore, implantation of this function is optional and depends on SY implementation.

#### **6.4.1.3 Time & Statistic Calls**

Functions in this group provide the most common operations with time and random numbers.



#### 6.4.1.3.1 syGetTimeInSec

##### Prototype

```
NQ_TIME syGetTimeInSec (  
    void  
);
```

##### Description

Get system time in POSIX format

##### Parameters

None

##### Return Value

The number of seconds elapsed since Jan 1, 1970

##### Notes

Return (-1) if your platform does not support time

#### 6.4.1.3.2 syGetTimeZone

##### Prototype

```
NQ_INT syGetTimeZone(  
    void  
);
```

##### Description

Get time offset

##### Parameters

None

##### Return Value

The number of minutes to be added to the local time to get GMT.  
This number is negative for GMT+ and positive for GMT-

##### Notes

#### 6.4.1.3.3    syGetTimeInMsec

##### Prototype

```
NQ_TIME syGetTimeZone(  
    void  
);
```

##### Description

Get system time in POSIX format (epoch) in milliseconds

##### Parameters

None

##### Return Value

The number of milliseconds elapsed since Jan 1, 1970 in milliseconds

##### Notes

#### 6.4.1.3.4 syConvertTimeSpecToTimeInMsec

## Prototype

```

NQ_TIME syConvertTimeSpecToTimeInMsec(
    void * val
);

```

## Description

Covert the **timespec** to time in milliseconds

## Parameters

Val	IN
	Time in a <b>&lt;b&gt;timespec&lt;/b&gt;</b> format

### Return Value

The converted time from `<b>timespec</b>` to a time in milliseconds

## Notes

6.4.1.3.5    `syConvertTimeInMsecToSec`

## Prototype

```
NQ_UINT32 syConvertTimeInMsecToSec(  
    NQ_TIME timeMsec  
);
```

## Description

Receive time in milliseconds (64 bit) and convert it to time in seconds (32 bit)

## Parameters

<code>timeMsec</code>	IN
Time in milliseconds resolution	

## Return Value

Time in seconds resolution

## Notes

## 6.4.1.3.6 syDecomposeTime

## Prototype

```
void syDecomposeTime(  
    NQ_TIME time  
    SYTimeFragments *decomposed  
);
```

## Description

Decompose POSIX time into time fragments. This function takes POSIX times and split it into year, month, day, hour, minute and second portions.

## Parameters

time	IN
The number of seconds elapsed since Jan 1, 1970	
decomposed	OUT
Pointer to a time fragments structure (see SYCOMMON.H). This structure will be filled with time fragments	

## Return Value

None

## Notes

## 6.4.1.3.7    syComposeTime

## Prototype

```
NQ_TIME syComposeTime(  
    const SYTimeFragments *decomposed  
);
```

## Description

Compose POSIX time from time fragments

## Parameters

decomposed                      IN  
                                    Pointer to a time fragments structure (see  
                                    SYCOMMON.H)

## Return Value

The number of seconds elapsed since Jan 1, 1970

## Notes

## 6.4.1.3.8 sySetRand

## Prototype

```
void sySetRand(  
    void  
);
```

## Description

Prepare random number generator.

## Parameters

None

## Return Value

None

## Notes

Default implementation uses current time as a seed



## 6.4.1.3.9 syRand

## Prototype

```
NQ_INT syRand(  
    void  
);
```

## Description

Random number generator.

## Parameters

None

## Return Value

Pseudo-random number in range 0 – RAND\_MAX

## Notes

sySetRand() should be called before

## 6.4.1.3.10 sySleep

## Prototype

```
void sySleep(  
    NQ_TIME secs  
);
```

## Description

Stop current thread for a number of seconds.

## Parameters

secs	IN
Number of seconds to wait	

## Return Value

None.

## Notes

#### 6.4.1.4 Threads

This group provides thread operations. Here, *thread* means a process that shares address space with the process by which this thread has been created. We distinguish between two types of threads:

- An *internal thread* is a thread created and executed by NQ code. NQ Client creates one thread of this kind – the thread that receives SMB responses. NQ Client creates this thread on startup and terminates it on shutdown.
- An *external thread* belongs to an application using NQ Client API. Multiple external threads may run in the same address space. NQ Client implicitly registers an external thread when it issues its first SMB request. On shutdown, NQ Client will release all resources, associated with external threads.

## 6.4.1.4.1 SYThread

## Prototype

```
SYThread <variable>;
```

## Description

Thread handle

## Notes

This type designates an abstract thread handle which NQ passes to the functions on this group. The definition of this type is platform-specific.

## 6.4.1.4.2 syThreadStart

## Prototype

```
void syThreadStart(  
    SYThread *threadHandlePtr,  
    void (*body)(void),  
    NQ_BOOL background  
);
```

## Description

Create an internal thread

## Parameters

threadHandlePtr	OUT
Pointer to a thread handle to create	
body	IN
Pointer to the function to be used as the thread body	
background	IN
TRUE if the thread should have low priority. FALSE if the thread should have normal priority.	

## Return Value

None

## Notes

The implementation of this call should create thread in a platform-specific manner. This call should start execution of the new thread by calling the *body* function. The thread should shutdown when the *body* function returns.

## 6.4.1.4.3 syIsValidThread

## Prototype

```
NQ_BOOL syIsValidThread(  
    SYThread threadHandle  
);
```

## Description

Check thread handle

## Parameters

threadHandle                      IN  
                                    Handle of thread to check

## Return Value

TRUE when the given handle represents a valid thread, FALSE – otherwise

## Notes

This call will be used in the future versions. Currently it may always return TRUE.

## 6.4.1.4.4    syThreadGetCurrent

## Prototype

```
SYThread syThreadGetCurrent (
    void
);
```

## Description

Get handle of the current thread

## Parameters

None

## Return Value

Handle to the thread in which context this call was issued.

## Notes

## 6.4.1.4.5 syThreadDestroy

## Prototype

```
void syThreadDestroy(  
    SYThread threadHandle  
);
```

## Description

Terminate thread

## Parameters

threadHandle            IN  
                          Handle of the thread to destroy

## Return Value

None

## Notes

This call should remove the given thread from system and release its resources.



## 6.4.1.4.6 SY\_THREADSET

## Prototype

```
void SY_THREADSET(  
    void * param  
);
```

## Description

Set a per-thread pointer

## Parameters

param	IN
A pointer to save on per-thread basis	

## Return Value

None

## Notes

This call must be implemented as a macro. It may be not defined. In this case the call 6.4.1.4.7 must not be defined as well. If both calls are not defined, NQ will not use per-thread pointers.

#### 6.4.1.4.7 SY\_THREADGET

##### Prototype

```
void * SY_THREADGET (  
    void  
);
```

##### Description

Get a per-thread pointer

##### Parameters

None

##### Return Value

A per-thread pointer as saved in 6.4.1.4.6.

##### Notes

This call must be implemented as a macro. It may be not defined. In this case the call 6.4.1.4.6 must not be defined as well. If both calls are not defined, NQ will not use per-thread pointers.

#### 6.4.1.5 Semaphores

This group provides semaphore operations. We use two types of semaphores:

- Mutual exclusion semaphores (*mutex*) guarantee that only one process (task) at a time has access to a critical resource guarded by this semaphore. A pair of mutex operations – *take* and *give* designates a critical section in the code.
- Binary semaphores provided access to a pool of limited resources – buffers. Up to  $n$  processes (tasks) may take a buffer from a pool until the next requestor will be blocked.

#### 6.4.1.5.1 SYMutex

##### Prototype

```
SYMutex <variable>;
```

##### Description

Mutual exception semaphore

##### Notes

We use mutex semaphores.

## 6.4.1.5.2 syMutexCreate

## Prototype

```
void syMutexCreate(  
    SYMutex *semaphore  
);
```

## Description

Create a mutual exception semaphore

## Parameters

semaphore	OUT
Pointer to a semaphore variable to create	

## Return Value

None

## Notes

## 6.4.1.5.3 syMutexDelete

## Prototype

```
void syMutexDelete(  
    SYMutex *semaphore  
);
```

## Description

Dispose a mutual exception semaphore

## Parameters

semaphore	IN
Pointer to a semaphore variable to dispose	

## Return Value

None

## Notes

Implementation of this call is system-dependent. On some operating systems there is no need of disposing mutexes. In such a case, this call may be a dummy macro.

#### 6.4.1.5.4 syMutexTake

##### Prototype

```
void syMutexTake(  
    SYMutex *semaphore  
);
```

##### Description

Lock a resource protected by a mutex

##### Parameters

semaphore	IN/OUT
Pointer to a mutex semaphore	

##### Return Value

None

##### Notes

This call is time-critical and is likely to be implemented as a macro

## 6.4.1.5.5 syMutexGive

## Prototype

```
void syMutexGive(  
    SYMutex *semaphore  
);
```

## Description

Unlock a resource protected by a mutex

## Parameters

semaphore	IN/OUT
Pointer to a mutex semaphore	

## Return Value

None

## Notes

This call is time-critical and likely to be implemented as a macro



#### 6.4.1.5.6   SYSemaphore

##### Prototype

```
SYSemaphore <variable>;
```

##### Description

Binary semaphore

##### Notes

We use binary semaphores to count limited resources.

## 6.4.1.5.7 sySemaphoreCreate

## Prototype

```
NQ_STATUS sySemaphoreCreate(  
    SYSemaphore *semaphore,  
    NQ_INT count  
);
```

## Description

Create a binary semaphore

## Parameters

semaphore	OUT
Pointer to a semaphore variable to create	
count	IN
Number of recourses to allocate	

## Return Value

NQ\_SUCCESS if the operation succeeded, NQ\_FAIL if an error occurred

## Notes

## 6.4.1.5.8 sySemaphoreDelete

## Prototype

```
void sySemaphoreDelete(  
    SYSemaphore semaphore  
);
```

## Description

Dispose a binary semaphore

## Parameters

semaphore                      IN  
semaphore variable to dispose

## Return Value

None

## Notes

## 6.4.1.5.9 sySemaphoreTake

## Prototype

```
void sySemaphoreTake(  
    SYSemaphore semaphore  
);
```

## Description

Lock a resource protected by a binary semaphore

## Parameters

semaphore                      IN/OUT  
Binary semaphore

## Return Value

None

## Notes

This call is time-critical and is likely to be implemented as a macro

## 6.4.1.5.10 sySemaphoreGive

## Prototype

```
void sySemaphoreGive(  
    SYSemaphore semaphore  
);
```

## Description

Unlock a resource protected by a binary semaphore

## Parameters

semaphore                      IN/OUT  
Binary semaphore

## Return Value

None

## Notes

This call is time-critical and is likely to be implemented as a macro

## 6.4.1.5.11 sySemaphoreResetCounter

## Prototype

```
void sySemaphoreResetCounter(  
    SYSemaphore* pSemID  
);
```

## Description

Set the semaphore counter to zero

## Parameters

pSemID	IN
Pointer to a semaphore variable	

## Return Value

None

## Notes

None

#### 6.4.1.5.12 SY\_SEMAPHORE\_AVAILABLE

##### Prototype

`SY_SEMAPHORE_AVAILABLE`

##### Description

System support for semaphores

##### Notes

When this parameter is defined, NQ assumes that the underlying operating system supports semaphores and the functions 6.4.1.5.6 to 6.4.1.5.10 are available. If this parameter is commented, NQ uses alternative algorithms (internally simulates semaphores) and avoids using those functions.

## 6.4.1.5.13 sySemaphoreTimedTake

## Prototype

```
NQ_INT sySemaphoreTimedTake(  
    SYSemaphore *sem,  
    NQ_INT timeout  
);
```

## Description

Lock a resource protected by a binary semaphore

## Parameters

sem	IN/OUT
	Binary semaphore
timeout	
	Timeout in seconds to wait for lock

## Return Value

NQ\_SUCCESS when the resource was locked or NQ\_FAIL on error

## Notes



#### **6.4.1.6 Networking**

This group defines function prototypes for socket operations. On platforms with BSD socket support the functions in this group may be implemented by means of the appropriate BSD functions.

#### 6.4.1.6.1 SY\_LOCALHOSTIP4

##### Prototype

SY\_LOCALHOSTIP4

##### Description

Definition of the "loopback" address for IPv4

##### Notes

The value of "self-address" for the target OS. Possible values are: 0x7F000001 (127, 0, 0, 1) or zero. This value should be defined in Network Byte Order.

#### 6.4.1.6.2 SY\_LOCALHOSTIP6

##### Prototype

`SY_LOCALHOSTIP6`

##### Description

Definition of the "loopback" address for IPv6

##### Notes

The value of "self-address" for the target OS. Possible values are: {0, 0, 0, 0, 0, 0, 0, 0x100) or eight zeros. This value should be defined in Network Byte Order.

#### 6.4.1.6.3   SY\_LINKLOCALIP

##### Prototype

SY\_ LINKLOCALIP

##### Description

Bit mask for “link-local” IPv6 addresses

##### Notes

The value of "link-local" bit mask for the target OS. Applicable to IPv6 addresses only. This value should be defined in Network Byte Order.

## 6.4.1.6.4 SY\_ANYIP4

## Prototype

SY\_ANYIP4

## Description

Definition of “any” IPv4 address

## Notes

The value of "any" IPv4 address for the target OS. Possible value is zero or INADDR\_ANY. This value should be defined in Network Byte Order.

## 6.4.1.6.5   SY\_ANYIP6

## Prototype

`SY_ANYIP6`

## Description

Definition of “any” IPv6 address

## Notes

The value of "any" IPv6 address for the target OS. Possible value is zero or IN6ADDR\_ANY. This value should be defined in Network Byte Order.

#### 6.4.1.6.6 SY\_INTERNALSOCKETPOOL

##### Prototype

SY\_INTERNALSOCKETPOOL

##### Description

Whether NQ uses global socket pool for internal communications

##### Notes

NQ uses internal (loop-back) sockets for communications between NQ Client from one side and NetBIOS on other side.

Normally, when this parameter defined, these sockets are created and bound once during NQ start. Then, all tasks share the same global pool of sockets.

Some operating systems support per-task sockets only, thus, does not allow sharing global sockets between tasks (e.g., pSOS). For such operating systems this parameter should be masked (commented).

#### 6.4.1.6.7   SYSocketHandle

##### Prototype

```
SYSocketHandle <variable>;
```

##### Description

System-independent socket handle

##### Notes

For BSD-compatible sockets this type may be substituted with *int*.



## 6.4.1.6.8    SYSocketSet

## Prototype

```
SYSocketSet <variable>;
```

## Description

System-independent socket set to be use in a *select* operation

## Notes

For BSD-compatible sockets this type may be substituted with *fd\_set*.

## 6.4.1.6.9 syIsValidSocket

## Prototype

```
NQ_BOOL syIsValidSocket(  
    SYSocketHandle socket  
);
```

## Description

Check whether a given socket is a valid socket

## Parameters

socket	IN
Socket to check	

## Return Value

TRUE when the socket handle is valid and FALSE when it is not a valid socket handle

## Notes

It is up to a developer to define which socket handle value is invalid. For BSD-compatible sockets this value is -1. This time-critical call is likely to be implemented as a macro

## 6.4.1.6.10 syInvalidSocket

## Prototype

```
SYSocketHandle syInvalidSocket(  
    void  
);
```

## Description

Generate an invalid socket handle

## Parameters

None

## Return Value

An invalid socket handle

## Notes

It is up to a developer to define which socket handle value is invalid. For BSD-compatible sockets this value is -1. This time-critical call is likely to be implemented as a macro

## 6.4.1.6.11 syIsSocketAlive

## Prototype

```
NQ_BOOL syIsSocketAlive(  
    SYSocketHandle socket  
);
```

## Description

Check socket availability for TCP operations.

## Parameters

socket	IN
Socket to check	

## Return Value

TRUE when the socket is available and connected FALSE when it is not accessible.

## Notes

It is up to a developer to determine a method of checking the socket. A possible solution is calling *select()* on this socket with zero timeout. For most TCP implementations this call will return zero for a valid socket.

## 6.4.1.6.12 syCreateSocket

## Prototype

```
SYSocketHandle syCreateSocket(  
    NQ_BOOL stream,  
    NQ_UINT familiy  
);
```

## Description

Create a new socket

## Parameters

stream	IN
	TRUE for TCP socket, FALSE for UDP socket
family	IN
	4 for IPv4, 6 for IPv6

## Return Value

New socket handle if the operation succeeded, illegal handle otherwise

## Notes

## 6.4.1.6.13 syAddSocketToSet

## Prototype

```
void syAddSocketToSet (
    SYSocketHandle socket,
    SYSocketSet *set
);
```

## Description

Add socket to a set of sockets to be used in a *select* operation

## Parameters

socket	IN
	Handle of the socket to be added to the set of sockets
set	OUT
	Pointer to the socket set

## Return Value

None

## Notes

This time-critical call is likely to be implemented as a macro

## 6.4.1.6.14 syRemoveSocketFromSet

## Prototype

```
void syRemoveSocketFromSet(  
    SYSocketHandle socket,  
    SYSocketSet *set  
);
```

## Description

Remove socket from a set of sockets

## Parameters

socket	IN
	Handle of the socket to be removed from the set of sockets
set	OUT
	Pointer to the socket set

## Return Value

None

## Notes

This time-critical call is likely to be implemented as a macro

## 6.4.1.6.15 syIsSocketSet

## Prototype

```
NQ_BOOL syIsSocketSet(  
    SYSocketHandle socket,  
    SYSocketSet *set  
);
```

## Description

Find socket in a set of sockets

## Parameters

socket	IN
	Handle of the socket to be find
set	IN
	Pointer to the socket set

## Return Value

TRUE if the socket is in the set, FALSE otherwise

## Notes

This time-critical call is likely to be implemented as a macro



## 6.4.1.6.16 syClearSocketSet

## Prototype

```
void syClearSocketSet(  
    SYSocketSet *set  
);
```

## Description

Remove all sockets from the set

## Parameters

set	OUT
Pointer to the socket set	

## Return Value

None

## Notes

This time-critical call is likely to be implemented as a macro

## 6.4.1.6.17 syShutdownSocket

## Prototype

```
NQ_STATUS syShutdownSocket(  
    SYSocketHandle socket  
);
```

## Description

Cancel all operation on socket

## Parameters

socket	IN
Socket handle	

## Return Value

NQ\_SUCCESS if the operation succeeded, NQ\_FAIL if an error occurred

## Notes

This operation is expected to disconnect a connected socket. On a UDP this operation should only invalidate the socket

## 6.4.1.6.18 syCloseSocket

## Prototype

```
NQ_STATUS syCloseSocket(  
    SYSocketHandle socket  
);
```

## Description

Release all resources used by the socket and release the handle

## Parameters

socket	IN
Socket handle	

## Return Value

NQ\_SUCCESS if the operation succeeded, NQ\_FAIL if an error occurred

## Notes

## 6.4.1.6.19 syListenSocket

## Prototype

```
NQ_STATUS syListenSocket(  
    SYSocketHandle socket,  
    NQ_INT backlog  
);
```

## Description

Start listening on a socket

## Parameters

socket	IN
Socket handle	
backlog	IN
Depth of the connection stack – maximum number of outstanding connection requests	

## Return Value

NQ\_SUCCESS if the operation succeeded, NQ\_FAIL if an error occurred

## Notes

## 6.4.1.6.20 sySelectSocket

## Prototype

```
NQ_INT sySelectSocket(  
    SYSocketSet *pSet,  
    NQ_UINT32 timeout  
);
```

## Description

Poll sockets in the socket set for incoming data

## Parameters

pSet	IN
	Pointer to the socket set containing sockets to poll
timeout	IN
	Timeout in seconds

## Return Value

Positive value - number of sockets with data pending, zero – timeout occurred, NQ\_FAIL if an error occurred

## Notes

#### 6.4.1.6.21 syAcceptSocket

##### Prototype

```
SYSocketHandle syAcceptSocket (
    SYSocketHandle socket,
    NQ_IPADDRESS *ip,
    NQ_PORT *port
);
```

##### Description

Accept connecting socket

##### Parameters

socket	IN	
		Listening socket
ip	OUT	
		Pointer to the buffer for IP address of the connecting socket. This buffer should be at least 4 bytes long. The value in it will be in Network Byte Order (NBO).
port	OUT	
		Pointer to the buffer for port number of the connecting socket. This buffer should be at least 2 bytes long. The value in it will be in Network Byte Order (NBO)

##### Return Value

New socket handle or invalid socket handle on error

##### Notes

## 6.4.1.6.22 syBindSocket

## Prototype

```
NQ_STATUS syBindSocket(  
    SYSocketHandle socket,  
    const NQ_IPADDRESS *ip,  
    NQ_PORT port  
);
```

## Description

Bind socket into IP address and port

## Parameters

socket	IN
	Handle of the socket to bind
ip	IN
	Pointer to IP address for the socket. This value should be in Network Byte Order (NBO).
port	IN
	Port number address for the socket. This value should be in Network Byte Order (NBO)

## Return Value

NQ\_SUCCESS if the operation completed, NQ\_FAIL on error

## Notes

This function should analyze the IP address type (either IPV4 or IPv6) and builds IP address in the system format (e.g., - sockaddr\_in/sockaddr\_in6) accordingly before binding the socket.





## 6.4.1.6.24 sySetClientSocketOptions

## Prototype

```
void sySetClientSocketOptions(  
    SYSocketHandle socket  
);
```

## Description

Set socket options for an accepted (client) socket

## Parameters

socket	IN
Socket handle	

## Return Value

None

## Notes

This operation may be implemented by means of standard socket options (like `SO_LINGER`, for instance). However, the optimum set of these options is platform-specific.

## 6.4.1.6.25 sySetDatagramSocketOptions

## Prototype

```
void sySetDatagramSocketOptions(  
    SYSocketHandle socket  
);
```

## Description

Set socket options for a UDP socket

## Parameters

socket	IN
Socket handle	

## Return Value

None

## Notes

This operation may be implemented by means of standard socket options (like `SO_LINGER`, for instance). However, the optimum set of these options is platform-specific.

## 6.4.1.6.26 sySetStreamSocketOptions

## Prototype

```
void sySetStreamSocketOptions(  
    SYSocketHandle socket  
);
```

## Description

Set socket options for a TCP socket (either server socket or a connected socket)

## Parameters

socket	IN
Socket handle	

## Return Value

None

## Notes

This operation may be implemented by means of standard socket options (like SO\_LINGER, for instance). However, the optimum set of these options is platform-specific.

#### 6.4.1.6.27 syGetSocketPortAndIP

##### Prototype

```
void syGetSocketPortAndIP(
    SYSocketHandle socket,
    NQ_IPADDRESS *ip,
    NQ_PORT *port
);
```

##### Description

Determine on what the given socket is bound

##### Parameters

socket	IN	
		Socket handle
ip	OUT	
		Pointer to a buffer for IP address. This buffer should be at least 4 bytes long for IPv4 and it should be at least 16 bytes long for IPv6. On return this buffer contains the address this socket is bound on. This value is in Network Byte Order (NBO)
port	OUT	
		Pointer to a buffer for port number. This buffer should be at least 2 bytes long. On return this buffer contains the port this socket is bound on. This value is in Network Byte Order (NBO)

##### Return Value

None

##### Notes

If the socket is not bound the result of this operation is undefined

#### 6.4.1.6.28 syConnectSocket

##### Prototype

```
NQ_STATUS syConnectSocket (
    SYSocketHandle socket,
    const NQ_IPADDRESS *ip,
    NQ_PORT port
);
```

##### Description

Connect to a remote server

##### Parameters

socket	IN	Connecting socket handle
ip	IN	Pointer to the IP address of the server. This value should be in Network Byte Order (NBO).
port	IN	Port number the server is listening. This value should be in Network Byte Order (NBO).

##### Return Value

NQ\_SUCCESS if the operation succeeded, NQ\_FAIL if an error occurred

##### Notes

This function should analyze the IP address type (either IPV4 or IPv6) and builds IP address in the system format (e.g., - sockaddr\_in/sockaddr\_in6) accordingly before binding the socket.

#### 6.4.1.6.29 sySendMulticast

##### Prototype

```
NQ_STATUS sySendMulticast(
    SYSocketHandle socket,
    const NQ_BYTE *buf,
    NQ_COUNT len,
    const NQ_IPADDRESS *ip,
    NQ_PORT port
);
```

##### Description

Send datagram to remote socket as a multicast

##### Parameters

socket	IN	
		Sending socket handle
buf	IN	
		Pointer to the data to be sent
len	IN	
		Length of the data to be sent
ip	IN	
		Pointer to the IP address to send the datagram to. This value should be a multicast address in Network Byte Order (NBO).
port	IN	
		Port number to send the datagram to. This value should be in Network Byte Order (NBO).

##### Return Value

The number of bytes actually sent or NQ\_FAIL if an error occurred

##### Notes

This function should perform platform-specific actions to allow sending multicast over the given socket. Then it should analyze the IP address type (either IPV4 or IPv6) and builds IP address in the system format (e.g., - sockaddr\_in/sockaddr\_in6) accordingly before sending datagram over the socket

## 6.4.1.6.30 sySendSocket

## Prototype

```
NQ_INT sySendSocket(  
    SYSocketHandle socket,  
    const NQ_BYTE *buf,  
    NQ_COUNT len  
);
```

## Description

Send data over a connected socket

## Parameters

socket	IN
Sending socket handle	
buf	IN
Pointer to the data to be sent	
len	IN
Length of the data to be sent	

## Return Value

The number of bytes actually sent or NQ\_FAIL if an error occurred

## Notes

#### 6.4.1.6.31 sySendSocketAsync

##### Prototype

```
NQ_STATUS sySendSocketAsync (
    SYSocketHandle socket,
    const NQ_BYTE *buf,
    NQ_COUNT len,
    void (*releaseFunc) (const NQ_BYTE *)
);
```

##### Description

Send data asynchronously over a connected socket

##### Parameters

socket	IN
Sending socket handle	
buf	IN
Pointer to the data to be sent	
len	IN
Length of the data to be sent	
releaseFunc	IN
callback function for releasing the buffer	

##### Return Value

The number of bytes queued or NQ\_FAIL if an error occurred

##### Notes

This function queues the buffer to asynchronous processing and immediately returns. When all bytes are sent, this function calls *releaseFunc()* passing *buf* as its only parameter. The *releaseFunc()* parameter has the same value on subsequent calls of this function that allows to save its value once in a static variable. NQ uses this function only when UD\_NS\_ASYNCSEND parameter is defined (see 7.3.2.6). Implementing this function requires asynchronous socket operations on the OS level.



#### 6.4.1.6.32 sySendToSocket

##### Prototype

```
NQ_INT sySendToSocket (
    SYSocketHandle socket,
    const NQ_BYTE *buf,
    NQ_COUNT len,
    const NQ_IPADDRESS *ip,
    NQ_PORT port
);
```

##### Description

Send datagram to remote socket on a particular computer

##### Parameters

socket	IN	
		Sending socket handle
buf	IN	
		Pointer to the data to be sent
len	IN	
		Length of the data to be sent
ip	IN	
		Pointer to the IP address to send the datagram to. This value should be in Network Byte Order (NBO).
port	IN	
		Port number to send the datagram to. This value should be in Network Byte Order (NBO).

##### Return Value

The number of bytes actually sent or NQ\_FAIL if an error occurred

##### Notes

This function should analyze the IP address type (either IPV4 or IPv6) and builds IP address in the system format (e.g., -sockaddr\_in/sockaddr\_in6) accordingly before sending datagram over the socket

#### 6.4.1.6.33 syRecvFromSocket

##### Prototype

```
NQ_INT syRecvFromSocket (
    SYSocketHandle socket,
    NQ_BYTE *buf,
    NQ_COUNT len,
    NQ_IPADDRESS *ip,
    NQ_PORT *port
);
```

##### Description

Receive datagram on socket

##### Parameters

socket	IN	
		Socket handle
buf	OUT	
		Pointer to the buffer for receiving data
len	IN	
		This buffer size
ip	OUT	
		Pointer to the buffer for IP address of the sender. This value is in Network Byte Order (NBO).
port	OUT	
		Pointer to the buffer for port number of the sender. This value is in Network Byte Order (NBO).

##### Return Value

The number of bytes received or NQ\_FAIL if an error occurred

##### Notes

## 6.4.1.6.34 syRecvSocket

## Prototype

```
NQ_INT syRecvSocket(  
    SYSocketHandle socket,  
    NQ_BYTE *buf,  
    NQ_COUNT len  
);
```

## Description

Receive data over socket

## Parameters

socket	IN
Socket handle	
buf	OUT
Pointer to the buffer for receiving data	
len	IN
This buffer size	

## Return Value

The number of bytes received or NQ\_FAIL if an error occurred

## Notes

## 6.4.1.6.35 syRecvSocketWithTimeout

## Prototype

```
NQ_INT syRecvSocketWithTimeout(  
    SYSocketHandle socket,  
    NQ_BYTE *buf,  
    NQ_COUNT len,  
    NQ_TIME secs,  
);
```

## Description

Receive data over a socket or time out

## Parameters

socket	IN
Socket handle	
buf	OUT
Pointer to the buffer for receiving data	
len	IN
This buffer size	
secs	IN
Timeout in seconds	

## Return Value

The number of bytes received or NQ\_FAIL if an error occurred. On timeout this call should return a zero value.

## Notes

The socket may be either a stream socket or a datagram socket

## 6.4.1.6.36 syGetDnsParams

## Prototype

```
void syGetDnsParams(  
    NQ_CHAR *domain,  
    NQ_IPADDRESS *server  
);
```

## Description

Determine parameters for DNS calls

## Parameters

domain	OUT
	Buffer for full qualified domain name like myDomain.Internal or myDomain.org
server	OUT
	Buffer for DNS IP address. This address may be either IPv4 or IPv6 address.

## Return Value

Data type NQ\_IPADDRESS is defined in

## Notes

## 6.4.1.6.37 syGetIPv6ScopeId

## Prototype

```
NQ_INT syGetIPv6ScopeId(  
    const NQ_IPADDRESS6 *ip  
);
```

## Description

Determine scope ID for by IPv6 address

## Parameters

ip	IN
IPv6 address for adapter	

## Return Value

IPv6 scope ID

## Notes

## 6.4.1.6.38 sySubscribeToMulticast

## Prototype

```
void sySubscribeToMulticast(  
    SYSocketHandle socket,  
    const NQ_IPADDRESS *ip  
);
```

## Description

Subscribe IP address to remote socket as a multicast

## Parameters

socket	IN
	Remote socket handle
ip	IN
	Pointer to the IP address to subscribe. This value should be a multicast address in Network Byte Order (NBO).

## Return Value

None

## Notes

This function should analyze the IP address type (either IPV4 or IPv6) and builds IP address in the system format (e.g., - sockaddr\_in/sockaddr\_in6) accordingly before subscribe it.

#### **6.4.1.7 Tasks**

These functions operate with tasks. NQ uses Process ID (PID) to uniquely identify a task. This value should be a number. We assume that in any OS there is a unique task identifier which is either a number or may be uniquely mapped on a number.



## 6.4.1.7.1 syGetPid

## Prototype

```
NQ_UINT syGetPid(  
    void  
);
```

## Description

Get ID of the current task

## Parameters

None

## Return Value

A unique ID of the current task

## Notes

NQ platform-independent software uses this number for identification purposes and does use it for task control. This number should be unique among all tasks running on this computer.

#### **6.4.1.8 Directories**

This group of functions operates with directories.

## 6.4.1.8.1   SYDirectory

## Prototype

`SYDirectory <variable>`

## Description

Directory handle

## Notes

This type designated a handle of an opened directory. NQ keeps it into internal database and passes it to other functions of this group.

## 6.4.1.8.2 syInvalidateDirectory

## Prototype

```
void syInvalidateDirectory(  
    SYDirectory *pDir  
);
```

## Description

Make directory handle invalid

## Parameters

pDir	OUT
	Pointer to the directory handle

## Return Value

None

## Notes

## 6.4.1.8.3 syIsValidDirectory

## Prototype

```
NQ_BOOL syIsValidDirectory(  
    SYDirectory dir  
);
```

## Description

Check if the given handle designates a valid directory

## Parameters

dir	OUT
Directory handle to check	

## Return Value

TRUE if the handle is valid, FALSE if it is invalid

## Notes

Since this function is time-critical it is likely to implement it as a macro.

## 6.4.1.8.4 syCreateDirectory

## Prototype

```
NQ_STATUS syCreateDirectory(  
    const NQ_WCHAR *path  
);
```

## Description

Create new directory

## Parameters

path	IN
Pointer to a string containing full path to a new directory	

## Return Value

NQ\_SUCCESS if the directory was created, NQ\_FAIL if an error occurred

## Notes

## 6.4.1.8.5    syDeleteDirectory

## Prototype

```
NQ_STATUS syDeleteDirectory(  
    const NQ_WCHAR *path  
);
```

## Description

Delete directory

## Parameters

path	IN
Pointer to a string containing full path to directory	

## Return Value

NQ\_SUCCESS if the directory was deleted, NQ\_FAIL if an error occurred

## Notes

## 6.4.1.8.6    syOpenDirectory

## Prototype

```
SYDirectory syOpenDirectory(  
    const NQ_WCHAR *path  
);
```

## Description

Open directory

## Parameters

path	IN
Pointer to a string containing full path to directory	

## Return Value

Handle for opened directory or invalid handle on error

## Notes



## 6.4.1.8.7 syFirstDirectoryFile

## Prototype

```
int syFirstDirectoryFile(  
    const NQ_WCHAR *path,  
    SYDirectory *pDir,  
    const NQ_WCHAR **fileName  
);
```

## Description

Open directory and read the first entry

## Parameters

path	IN
	Pointer to a string containing full path to directory
pDir	OUT
	Pointer to buffer for directory handle. After successful operation this buffer gets the handle of opened directory. This buffer should be big enough.
fileName	OUT
	Pointer to pointer to a filename. After successful operation this pointer will contain an address of the first file name. For an empty directory this pointer will be NULL.

## Return Value

Success if the operation succeeded, NQ\_FAIL if an error occurred

## Notes

## 6.4.1.8.8 syNextDirectoryFile

## Prototype

```
NQ_STATUS syNextDirectoryFile(  
    SYDirectory dir,  
    const NQ_WCHAR **fileName  
);
```

## Description

Read next entry from an opened directory

## Parameters

Dir	IN
Handle of an opened directory.	
Filename	OUT
Pointer to pointer to a filename. After successful operation this pointer will contain an address of the first file name. On the end of directory scan, this pointer will be NULL.	

## Return Value

NQ\_SUCCESS if the operation succeeded, NQ\_FAIL if an error occurred

## Notes

## 6.4.1.8.9 syCloseDirectory

## Prototype

```
NQ_STATUS syCloseDirectory(  
    SYDirectory dir  
);
```

## Description

Closed directory handle

## Parameters

dir	IN
Handle of an opened directory.	

## Return Value

Success if the operation succeeded, NQ\_FAIL if an error occurred

## Notes

#### **6.4.1.9 Files**

This group of functions operates with files.

#### 6.4.1.9.1 SY\_UNICODEFILESYSTEM

Prototype

None

Description

Unicode support

Notes

This pre-processor variable defines whether the target file system supports Unicode file names. Without this support even if NQ software is generated with UD\_CM\_UNICODEAPPLICATION parameter (see 7.3.1.6) all file names are converted to ANSI prior to calling OS functions.

The line with this variable should be commented out when the target file system does not support Unicode.

#### 6.4.1.9.2 SY\_DRIVELETTERINPATH

Prototype

SY\_DRIVELETTERINPATH

Description

Whether the target file system requires drive letter in a file path

Notes

Define this parameter when path syntaxes in the target file system prefixes with a drive letter like “D:...”.

This parameter is used only with SRVSVC service for adding new share with Remote Management (see [1] ). Windows CIFS/SMB clients require this drive letter to be in the share path. NQ Server uses this parameter to decide whether to strip this drive letter out of the path (for Posix-style target file systems) or to leave it (for Windows-style target file systems). By default this parameter is omitted.

#### 6.4.1.9.3 SY\_PATHSEPARATOR

Prototype

None

Description

Path separator for the target file system

Notes

This pre-processor variable defines file path separator character ("/" for UNIX files systems and "\" for Windows-based file systems).

## 6.4.1.9.4   SY\_CP\_FIRSTILLEGALCHAR

## Prototype

None

## Description

A list of characters which are not valid as the first character in a file name.

## Notes

Some file-systems do not allow particular ANSI codes in a file name. NQ uses functions *cmAnsiToFs* and *cmFsToAnsi* (see 5.5.1 and 5.5.2) to replace such characters. This value defines those symbols that are illegal at the first position in a file name.



#### 6.4.1.9.5   SY\_CP\_ANYILLEGALCHAR

Prototype

None

Description

A list of characters which are not valid in a file name.

Notes

Some file-systems do not allow particular ANSI codes in a file name. NQ uses functions *cmAnsiToFs* and *cmFsToAnsi* (see 5.5.1 and 5.5.2) to replace such characters. This value defines those symbols that are illegal in any place in a file name.

#### 6.4.1.9.6   SY\_FS\_SEPARATEINFOMODE

Prototype

None

Description

When defined allows separate call for querying file information.

Notes

Defined on Windows platforms only.

## 6.4.1.9.7   SYFile

## Prototype

`SYFile <variable>`

## Description

File handle

## Notes

This type designated a handle of an opened file. NQ keeps it in the internal database and passes it to other functions of this group.

## 6.4.1.9.8 syInvalidateFile

## Prototype

```
void syInvalidateFile(  
    SYFile *pFile  
);
```

## Description

Set invalid value into file handle

## Parameters

pFile	OUT
Pointer to a file handle	

## Return Value

None

## Notes

## 6.4.1.9.9 syIsValidFile

## Prototype

```
NQ_BOOL syIsValidFile(  
    SYFile file  
);
```

## Description

Check whether file handle is valid

## Parameters

file	IN
File handle	

## Return Value

TRUE if the handle is valid, FALSE if it is an invalid handle

## Notes

Since this call is time-critical it is likely to be implemented as a macro.

## 6.4.1.9.10 syCreateFile

## Prototype

```
SYFile syCreateFile(  
    const NQ_WCHAR *name,  
    NQ_BOOL denyRead,  
    NQ_BOOL denyExecute,  
    NQ_BOOL denyWrite  
);
```

## Description

Create new file

## Parameters

name	IN
Name of the file to create	
denyRead	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring read access.	
denyExecute	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring execute access.	
denyWrite	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring write access.	

## Return Value

File handle on success or an invalid file handle on error

## Notes

Since this call is time-critical it is likely to be implemented as a macro.

## 6.4.1.9.11 syDeleteFile

## Prototype

```
NQ_STATUS syDeleteFile(  
    const NQ_WCHAR *name  
);
```

## Description

Delete a file

## Parameters

Name	IN
Name of the file to delete	

## Return Value

NQ\_SUCCESS if the file was deleted and NQ\_FAIL on error

## Notes

Since this call is time-critical it is likely to be implemented as a macro.

## 6.4.1.9.12 syRenameFile

## Prototype

```
NQ_STATUS syRenameFile(  
    const NQ_WCHAR *oldName,  
    const NQ_WCHAR *newName  
);
```

## Description

Rename a file

## Parameters

oldName	IN
Name of an existing file	
newName	IN
New name	

## Return Value

NQ\_SUCCESS if the file was renamed and NQ\_FAIL on error

## Notes

Since this call is time-critical it is likely to be implemented as a macro.



## 6.4.1.9.13 syOpenFileForRead

## Prototype

```
SYFile syOpenFileForRead(  
    const NQ_WCHAR *name,  
    NQ_BOOL denyRead,  
    NQ_BOOL denyExecute,  
    NQ_BOOL denyWrite  
);
```

## Description

Open file for reading

## Parameters

name	IN
Name of the file to open	
denyRead	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring read access.	
denyExecute	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring execute access.	
denyWrite	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring write access.	

## Return Value

File handle on success or invalid file handle on error

## Notes

Since this call is time-critical it is likely to be implemented as a macro.

## 6.4.1.9.14 syOpenFileForWrite

## Prototype

```
SYFile syOpenFileForWrite(  
    const NQ_WCHAR *name,  
    NQ_BOOL denyRead,  
    NQ_BOOL denyExecute,  
    NQ_BOOL denyWrite,  
);
```

## Description

Open file for writing

## Parameters

name	IN
Name of the file to open	
denyRead	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring read access.	
denyExecute	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring execute access.	
denyWrite	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring write access.	

## Return Value

File handle on success or invalid file handle on error

## Notes

Since this call is time-critical it is likely to be implemented as a macro.

## 6.4.1.9.15 syOpenFileForReadWrite

## Prototype

```
SYFile syOpenFileForReadWrite(  
    const NQ_WCHAR *name,  
    NQ_BOOL denyRead,  
    NQ_BOOL denyExecute,  
    NQ_BOOL denyWrite,  
);
```

## Description

Open file for reading and writing

## Parameters

name	IN
Name of the file to open	
denyRead	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring read access.	
denyExecute	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring execute access.	
denyWrite	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring write access.	

## Return Value

File handle on success or invalid file handle on error

## Notes

Since this call is time-critical it is likely to be implemented as a macro.

## 6.4.1.9.16 syOpenFileForInfo

## Prototype

```
SYFile syOpenFileForInfo(  
    const NQ_WCHAR *name,  
    NQ_BOOL denyRead,  
    NQ_BOOL denyExecute,  
    NQ_BOOL denyWrite,  
);
```

## Description

Open file for info

## Parameters

name	IN
Name of the file to open	
denyRead	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring read access.	
denyExecute	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring execute access.	
denyWrite	IN
This parameter defines whether to deny (TRUE) or to allow (FALSE) subsequent open operation on this file requiring write access.	

## Return Value

File handle on success or invalid file handle on error

## Notes

Since this call is time-critical it is likely to be implemented as a macro. Implemented for Windows platforms only, when SY\_FS\_SEPARATEINFOMODE defined (see 6.4.1.9.66.4.1.9.6).

## 6.4.1.9.17 syTruncateFile

## Prototype

```
NQ_STATUS syTruncateFile(  
    SYFile file,  
    NQ_UINT32 offLow,  
    NQ_UINT32 offHigh  
);
```

## Description

Open file for reading and writing

## Parameters

file	IN
	Handle of the file to truncate
offLow	IN
	Low 32 bits of the desired file size
offHigh	IN
	High 32 bits of the desired file size

## Return Value

NQ\_SUCCESS when the file was truncated or NQ\_FAIL on error

## Notes

This operation is only used for decreasing file size. When the desired file size is greater than its current size, this operation's result is undefined.

Since this call is time-critical it is likely to be implemented as a macro.

## 6.4.1.9.18 syFlushFile

## Prototype

```
NQ_STATUS syFlushFile(  
    SYFile file  
);
```

## Description

Synchronize cash memory with the media

## Parameters

file	IN
Handle of the file	

## Return Value

NQ\_SUCCESS when a file was synchronized or NQ\_FAIL on error

## Notes

Since this call is time-critical it is likely to be implemented as a macro.

## 6.4.1.9.19 syReadFile

## Prototype

```
int syReadFile(  
    SYFile file,  
    NQ_BYTE *buf,  
    NQ_COUNT len  
);
```

## Description

Read data from the current position in the file

## Parameters

file	IN
Handle of the file	
buf	OUT
Buffer for data	
len	IN
Number of bytes to read	

## Return Value

Number of bytes actually read from the file or NQ\_FAIL on error

## Notes

The buffer is big enough to store the required amount data.  
After this operation the file position should advance by the return value.

## 6.4.1.9.20 syWriteFile

## Prototype

```
NQ_INT syWriteFile(  
    SYFile file,  
    const NQ_BYTE *buf,  
    NQ_COUNT len  
);
```

## Description

Write data from the current position in the file

## Parameters

file	IN
Handle of the file	
buf	IN
Pointer to the data	
len	IN
Number of bytes to write	

## Return Value

Number of bytes actually written to the file or NQ\_FAIL on error

## Notes

After this operation the file position should advance by the return value.



## 6.4.1.9.21 syCloseFile

## Prototype

```
NQ_STATUS syCloseFile(  
    SYFile file  
);
```

## Description

Close opened file

## Parameters

file	IN
Handle of the file	

## Return Value

NQ\_SUCCESS if the file was close or NQ\_FAIL on error

## Notes

Since this operation is time-critical it is likely to implement it as a macro.

## 6.4.1.9.22 sySeekFileCurrent

## Prototype

```
NQ_INT32 sySeekFileCurrent(  
    SYFile file,  
    NQ_INT32 offLow,  
    NQ_INT32 offHigh  
);
```

## Description

Position file to a specified offset from the current position

## Parameters

file	IN
Handle of the file	
offLow	IN
Low 32 bits of the offset	
offHigh	IN
High 32 bits of the offset (should be ignored)	

## Return Value

NQ\_FAIL on error, new offset or NQ\_SUCCESS otherwise (see below)

## Notes

This call is used with offset values less of 32 bits only. Parameter *offHigh* is specified for compatibility only and should be ignored. Parameter *offLow* may specify a negative value. If the resulted file position is above the file end, the file should be extended.

## 6.4.1.9.23 sySeekFileStart

## Prototype

```
NQ_INT32 sySeekFileStart(  
    SYFile file,  
    NQ_INT32 offLow,  
    NQ_INT32 offHigh  
);
```

## Description

Position file to a specified offset from the beginning of the file

## Parameters

file	IN
Handle of the file	
offLow	IN
Low 32 bits of the offset	
offHigh	IN
High 32 bits of the offset	

## Return Value

NQ\_FAIL on error, new offset or NQ\_SUCCESS otherwise (see below)

## Notes

File systems with less than 4GB of per-file space should not use the last parameter. Such systems should consider offLow as a 31 bit value and return new offset in the file as the return value.

File systems with more than 4Gb per file should return NQ\_SUCCESS regardless of the resulted offset in the file. If the resulted file position is above the file end, the file should be extended.

## 6.4.1.9.24 sySeekFileEnd

## Prototype

```
NQ_INT32 sySeekFileEnd(  
    SYFile file,  
    NQ_INT32 offLow,  
    NQ_INT32 offHigh  
);
```

## Description

Position file to a specified offset from the end of the file

## Parameters

file	IN
Handle of the file	
offLow	IN
Low 32 bits of the offset	
offHigh	IN
High 32 bits of the offset	

## Return Value

NQ\_FAIL on error, new offset or NQ\_SUCCESS otherwise (see below)

## Notes

This call is used with offset values less of 32 bits only. Parameter *offHigh* is specified for compatibility only and should be ignored. Parameter *offLow* may specify a negative value. A positive value should extend the file.

## 6.4.1.9.25 syPrintf

## Prototype

```
NQ_INT syPrintf(  
    const NQ_CHAR *format,  
    ...  
);
```

## Description

Writes formatted string to stdout

## Parameters

format	IN
	Format string
...	IN
	Variable argument list

## Return Value

Number of characters written to stdout

## Notes

The synopsis of this call is exactly the same as for *printf()* function from *stdio.h*.

## 6.4.1.9.26 sySprintf

## Prototype

```
NQ_INT sySprintf(  
    NQ_CHAR *buffer,  
    const NQ_CHAR *format,  
    ...  
);
```

## Description

Writes formatted string to buffer

## Parameters

buffer	OUT
	Output string
format	IN
	Format string
...	IN
	Variable argument list

## Return Value

Number of characters written to the output string

## Notes

The synopsis of this call is exactly the same as for *sprintf()* function from *stdio.h*.

## 6.4.1.9.27 sySscanf

## Prototype

```
NQ_INT sySscanf(  
    const NQ_CHAR *inpStr,  
    const NQ_CHAR *format,  
    ...  
);
```

## Description

Parse string according to format

## Parameters

inpStr	IN
String to parse	
format	IN
Format string	
...	IN
Variable list of pointers to output buffers	

## Return Value

Number of items parsed

## Notes

The synopsis of this call is exactly the same as for *sscanf()* function from *stdio.h*.

## 6.4.1.9.28 syGetFileInformation

## Prototype

```
NQ_STATUS syGetFileInformation(  
    SYFile file,  
    const NQ_WCHAR *name  
    SYFileInformation *fileInfo  
);
```

## Description

Read file information providing file handle

## Parameters

file	IN
File handle	
name	IN
File name	
fileInfo	OUT
Buffer for file information.	

## Return Value

NQ\_SUCCESS if the information was read into the buffer or  
NQ\_FAIL on error

## Notes

This function is intended to obtain file information by a handle of an open file. This makes the second parameter (*name*) superfluous. Some operating systems, however, do not support this operation. In such case the second is used instead of file handle and this function becomes an equivalent of *syGetFileInformationByName* (see 6.4.1.9.29).

This function uses *SYFileInformation* structure to pass information to NQ



## 6.4.1.9.29 syGetFileInformationByName

## Prototype

```
NQ_STATUS syGetFileInformationByName(  
    const NQ_WCHAR *fileName,  
    SYFileInformation *fileInfo  
);
```

## Description

Read file information providing file name

## Parameters

filename	IN
Name of the file	
fileInfo	OUT
Buffer for file information.	

## Return Value

NQ\_SUCCESS if the information was read into the buffer or  
NQ\_FAIL on error

## Notes

This function uses SYFileInformation structure to pass information  
to NQ

## 6.4.1.9.30 sySetFileInformation

## Prototype

```
NQ_STATUS sySetFileInformation(  
    const NQ_WCHAR *fileName,  
    SYFile handle  
    const SYFileInformation *fileInfo  
);
```

## Description

Write file information

## Parameters

filename	IN
Name of the file	
handle	IN
File handle	
fileInfo	OUT
Buffer with file information.	

## Return Value

NQ\_SUCCESS if the information was written into the file or NQ\_FAIL on error

## Notes

This function modifies file information from a SYFileInformation structure provided by NQ. File can be designated by either file name or file handle. The recommended usage of those two parameters is as follows:

- If *syIsValidFile(handle)* (6.4.1.9.9) evaluates to *TRUE* - access file by handle (when applicable).
- Otherwise, access file by name

## 6.4.1.9.31 syGetVolumeInformation

## Prototype

```
NQ_STATUS syGetVolumeInformation(  
    const NQ_WCHAR *volumeName,  
    SYVolumeInformation *volumeInfo  
);
```

## Description

Read volume information

## Parameters

volumeName	IN
Name of the volume of interest	
volumeInfo	OUT
Buffer for volume information.	

## Return Value

NQ\_SUCCESS if the information was read into the buffer or  
NQ\_FAIL on error

## Notes

This function uses SYVolumeInformation structure to pass  
information to NQ

## 6.4.1.9.32 syGetSecurityDescriptor

## Prototype

```
int syGetSecurityDescriptor(  
    int file,  
    long fieldId,  
    void *buffer  
);
```

## Description

Read security descriptor

## Parameters

file	IN
File handle	
fieldId	IN
Specifies the field of interest in the security descriptor	
buffer	OUT
Buffer for the data	

## Return Value

Length of the data read into the buffer or NQ\_FAIL on error

## Notes

If the target file system does not support security descriptors the implementation should return NQ\_FAIL.

## 6.4.1.9.33 sySetSecurityDescriptor

## Prototype

```
int sySetSecurityDescriptor(  
    int file,  
    long fieldId,  
    const void *buffer  
);
```

## Description

Write security descriptor

## Parameters

file	IN
File handle	
fieldId	IN
Specifies the field of interest in the security descriptor	
buffer	IN
Pointer to the data to write	

## Return Value

NQ\_SUCCESS when the data was written or NQ\_FAIL on error

## Notes

If the target file system does not support security descriptors the implementation should return NQ\_FAIL.

DOS read only is represented in UNIX by removing everyone's write bit.  
 DOS archive is represented in UNIX by the user's execute bit.  
 DOS system is represented in UNIX by the group's execute bit.  
 DOS hidden is represented in UNIX by the other's execute bit.  
 DOS directory is represented in UNIX by UNIX 's directory bit.

## 6.4.1.9.35 syGetFileSize

## Prototype

```
NQ_STATUS syGetFileSize(  
    SYFile file,  
    NQ_UINT64 *size  
);
```

## Description

Read file size providing file handle

## Parameters

file	IN
	File handle
size	OUT
	Variable to hold file size

## Return Value

NQ\_SUCCESS if the information was read into the variable or  
NQ\_FAIL on error

## 6.4.1.9.36 syTraceCreateFile

## Prototype

```
SYFile syTraceCreateFile(  
    const NQ_WCHAR *name  
);
```

## Description

Create new trace file

## Parameters

name	IN
Name of the file to create	

## Return Value

File handle on success or an invalid file handle on error

## Notes

Use a different function for traces since YNQ internal trace mechanism run in a different thread, this will lead to a file corruption when using the same static variables with a different threads.



## 6.4.1.9.37 syTraceDeleteFile

## Prototype

```
NQ_STATUS syTraceDeleteFile(  
    const NQ_WCHAR *name  
);
```

## Description

Delete a trace file

## Parameters

Name	IN
Name of the file to delete	

## Return Value

NQ\_SUCCESS if the file was deleted and NQ\_FAIL on error

## Notes

Use a different function for traces since YNQ internal trace mechanism run in a different thread, this will lead to a file corruption when using the same static variables with a different threads.

#### **6.4.1.10      File Locking**

Functions of this group perform locking or unlocking of byte ranges inside an opened file. The two calls below are correspondent to lock/unlock operations respectively of the SMB\_COM\_LOCKING command. The implementation of these functions should follow CIFS/SMB recommendations for this command (see [3] ).

If the target file system does not support locking of byte ranges the implementation should return NQ\_SUCCESS for both of the calls below.

#### 6.4.1.10.1 syLockFile

##### Prototype

```
NQ_STATUS syLockFile(
    SYFile file,
    NQ_UINT32 offsetHigh,
    NQ_UINT32 offsetLow,
    NQ_UINT32 lengthHigh,
    NQ_UINT32 lengthLow,
    NQ_BYTE lockType,
    NQ_BYTE oplockLevel
);
```

##### Description

Lock a byte range in an opened file

##### Parameters

file	IN	
		File handle
offsetHigh	IN	
		High 32 bits of the range offset
offsetLow	IN	
		Low 32 bits of the range offset
lengthHigh	IN	
		High 32 bits of the range length
lengthLow	IN	
		Low 32 bits of the range length
lockType	IN	
		Lock type as specified by CIFS/SMB (see below)
oplockLevel	IN	
		Oplock level as specified by CIFS/SMB (see below)

##### Return Value

NQ\_SUCCESS when the byte range was locked or NQ\_FAIL on error

##### Notes

This call corresponds to the SMB\_COM\_LOCKING\_ANDX command. See [3] for more information about the parameters above and the implementation algorithm.

If the target file system does not support locking of byte ranges the implementation should return NQ\_SUCCESS.

#### 6.4.1.10.2 syUnlockFile

##### Prototype

```
NQ_STATUS syUnlockFile(
    SYFile file,
    NQ_UINT32 offsetHigh,
    NQ_UINT32 offsetLow,
    NQ_UINT32 lengthHigh,
    NQ_UINT32 lengthLow,
    NQ_UINT32 timeout
);
```

##### Description

Unlock a byte range in an opened file

##### Parameters

file	IN
File handle	
offsetHigh	IN
High 32 bits of the range offset	
offsetLow	IN
Low 32 bits of the range offset	
lengthHigh	IN
High 32 bits of the range length	
lengthLow	IN
Low 32 bits of the range length	
timeout	IN
Timeout in milliseconds to wait for unlock	

##### Return Value

NQ\_SUCCESS when the byte range was unlocked or NQ\_FAIL on error

##### Notes

This call corresponds to the SMB\_COM\_LOCKING\_ANDX command. See [3] for more information about the parameters above and the implementation algorithm.  
If the target file system does not support locking of byte ranges the implementation should return NQ\_SUCCESS.

#### **6.4.1.11      Networking**

This group contains several functions that require system-dependent or project-dependent information.

## 6.4.1.11.1 syGetHostName

## Prototype

```
void syGetHostName (  
    NQ_CHAR *nameBuffer,  
    NQ_UINT bufferSize  
);
```

## Description

Get the default computer name

## Parameters

nameBuffer	OUT
Pointer to the buffer	
bufferLength	IN
The buffer length	

## Return Value

None

## Notes

The computer name is a single-character string which does not have to include a null terminator.

## 6.4.1.11.2 syGetLastSmbError

## Prototype

```
NQ_UINT32 syGetLastSmbError(  
    void  
);
```

## Description

Convert last system error to SMB error code

## Parameters

None

## Return Value

SMB status code in DOS format (see Notes)

## Notes

The return value is a 32-bit integer values composed by the following formula:

$$\langle \text{return value} \rangle = 0x10000 * \langle \text{code} \rangle + \langle \text{class} \rangle$$

Where:

$\langle \text{class} \rangle$  - error class

$\langle \text{code} \rangle$  - error code

For error classes and error codes in DOS format see [3] .

This call is intended to withdraw the last system error and to convert it into the most appropriate SMB error.

Some implementations of this function delegate this call to `udGetSmbError()` (see 7.2.5.1).

The return value should be a non-zero value.

## 6.4.1.11.3 sySetLastNqError

## Prototype

```
void sySetLastNqError(  
    NQ_UINT32 nqErr  
);
```

## Description

Deprecated

## Parameters

nqErr	IN
Last NQ error	

## Return Value

None

## Notes

See also *sySetLastError()* (see 6.4.2.5.2).



#### 6.4.1.11.4 syGetAdapter

##### Prototype

```
NQ_STATUS syGetAdapter(
    NQ_INT idx,
    NQ_IPADDRESS4 *ip,
    NQ_IPADDRESS6 *ip6,
    NQ_IPADDRESS4 *subnet,
    NQ_IPADDRESS4 *wins
);
```

##### Description

Get information about a network adapter

##### Parameters

idx	IN	
		Zero-based adapter index
ip	OUT	
		Buffer for IPv4 address (should be at least four bytes long)
ip6	OUT	
		Buffer for IPv6 address (should be at least 16 bytes long)
subnet	OUT	
		Buffer for subnet mask address (should be at least four bytes long)
wins	OUT	
		Buffer for WINS address (should be at least four bytes long). If no WINS is specified for this interface, the buffer should be filled by four zero bytes.

##### Return Value

NQ\_SUCCESS if interface with the given index exists, NQ\_FAIL if there is no interface with such index or information cannot be obtained.

##### Notes

This function scans available network interfaces (adapters). NQ calls this function repeatedly, increasing *idx* until NQ\_FAIL is returned. The implementation of this function should answer to the following restrictions:

- Not active interfaces should be skipped
- Loop-back interface should be skipped if detected
- Interface with no broadcast capability should be skipped

This function's implementation should skip  $\langle idx - 1 \rangle$  interfaces that do not fall in the above categories. It should report on *idx*'s interface. When IPv6 transport is required, this function should find a pair of IPv4 and IPv6 addresses for the same adapter.

## 6.4.1.11.5 syGetMacAddress

## Prototype

```
void syGetMacAddress(  
    NQ_IPADDRESS4 ip,  
    NQ_BYTE *macBuffer  
);
```

## Description

Get MAC address by IP address

## Parameters

ip	IN
IP address	
macBuffer	OUT
Buffer for MAC address (is 6 bytes long)	

## Return Value

NQ\_SUCCESS if interface with the given index exists, NQ\_FAIL if there is no interface with such index or information cannot be obtained.

## Notes

While this function. Is mandatory, the information it provides serves display purposes only. Implementations that cannot withdraw MAC address may set the result buffer to six zeroes. The value provided by this function does not affect NQ functionality.

## 6.4.1.11.6 syGetHostByName

## Prototype

```
NQ_IPADDRESS4 syGetHostByName(  
    const char* name  
);
```

## Description

Find host IP by its name

## Parameters

name	IN
Host name	

## Return Value

Host IPv4 address (should be at least four bytes long).

## Notes

this function is used by client when  
UD\_NB\_INCLUDENAMESERVICE is not defined.

#### **6.4.1.12      SASL/GSSAPI/Kerberos Interface**

This group contains definitions, data types and calls for extended authentication mechanism over SASL/GSSAPI/Kerberos. The information in this section is relevant only when `UD_CC_INCLUDEEXTENDEDSECURITY` and `UD_CC_INCLUDEEXTENDEDSECURITY_KERBEROS` are defined in *udparams.h*. SASL/GSSAPI/Kerberos interface is defined in *sysasl.h*.

#### 6.4.1.12.1 SYSaslContext

##### Prototype

`SYSaslContext`

##### Description

SASL context handle

##### Notes

This is a type definition for SASL context. A handle is initially generated and returned by the *sySaslContextCreate()* call (see 6.4.1.12.8). It is used as an identification parameter in all subsequent calls.

## 6.4.1.12.2 SYSaslCallback

## Prototype

```
typedef void SYSaslCallback(  
    void* resource,  
    NQ_WCHAR* user,  
    NQ_WCHAR* password,  
    NQ_WCHAR* domain  
);
```

## Description

SASL credentials callback

## Parameters

resource	IN
	Pointer to the share path
user	IN
	Pointer to user name string
password	IN
	Pointer to the password string
domain	IN
	Pointer to domain name string

## Return Value

None

## Notes

This function is called by SASL when it requires user credentials. This happens in the context of a 6.4.1.12.11 or 6.4.1.12.12 call. The first parameter in this function call should be the same as passed in calls to 6.4.1.12.11 or 6.4.1.12.12.

All strings passed to this callback function are in NQ\_WCHAR characters. See [1] and [2] for how to convert ASCII or Unicode strings into NQ\_WCHAR strings.

## 6.4.1.12.3 sySaslContextIsValid

## Prototype

```
NQ_BOOL  
sySaslContextIsValid(  
    NQ_BYTE* context  
);
```

## Description

Validate SASL context

## Parameters

context	IN
Context to validate	

## Return Value

TRUE if this handle is valid, FALSE otherwise

## Notes

## 6.4.1.12.4 sySaslContextInvalidate

## Prototype

```
void  
sySaslContextInvalidate(  
    NQ_BYTE* context  
);
```

## Description

Set context to be invalid

## Parameters

context	IN
Context to make invalid	

## Return Value

None

## Notes



## 6.4.1.12.5 sySaslGetSecurityMechanism

## Prototype

```
const NQ_CHAR*  
syGetSaslSecurityMechanism(  
    void  
);
```

## Description

Get string ID of the security mechanism

## Parameters

None

## Return Value

Pointer to the string identification of the security mechanism installed in SASL.

## Notes

SPNEGO protocol by MS Windows series uses ASN1 identifiers for security mechanisms while SASL requires string identification. This function provides NQ with this identification. On any call, this function should return the same result.

## 6.4.1.12.6 sySaslClientInit

## Prototype

```
NQ_BOOL  
sySaslClientInit(  
    void* callback  
);
```

## Description

Initialize this module for client operations

## Parameters

callback	Function to be installed as the callback. See 6.4.1.12.2 for this callback prototype.
----------	---

## Return Value

TRUE on success, FALSE on error

## Notes

NQ Client calls this function once on initialization. The function provided should conform to 6.4.1.12.2.

## 6.4.1.12.7 sySaslClientStop

## Prototype

```
NQ_BOOL  
sySaslClientStop(  
    void  
);
```

## Description

Release resources used by this module

## Parameters

None

## Return Value

TRUE on success, FALSE on error

## Notes

NQ Client calls this function once on shutdown.

## 6.4.1.12.8 sySaslContextCreate

## Prototype

```
SYSaslContext  
sySaslContextCreate(  
    const NQ_CHAR* principal,  
    NQ_BOOL isSmb2,  
    NQ_BOOL signingOn,  
);
```

## Description

Create client-side context for security negotiation

## Parameters

<code>principal</code>	IN
Name of the server to connect	
<code>isSmb2</code>	IN
TRUE to use SMB2-style encryptions, FALSE to use SMB-style encryptions	
<code>signingOn</code>	IN
TRUE when authenticating with message signing intention, FALSE when message signing is not required	

## Return Value

New context or invalid context

## Notes

NQ Client calls this function on session start after getting Negotiate response. For signed SMB traffic Kerberos exchange should use a restricted list of crypters, which is achieved by specifying TRUE in the *restrict* parameter.

## 6.4.1.12.9 sySaslContextDispose

## Prototype

```
NQ_BOOL  
sySaslContextDispose(  
    NQ_BYTE* context  
);
```

## Description

Dispose unused context and all related resources

## Parameters

context	IN
Context to dispose	

## Return Value

TRUE on success, FALSE on failure

## Notes

After this call the context cannot be used anymore. If SASL is guarantied to dispose unused context, this call may be dummy.

6.4.1.12.10 `sySaslClientSetMechanism`

## Prototype

```
NQ_BOOL  
sySaslClientSetMechanism(  
    NQ_BYTE* context,  
    const NQ_CHAR* name  
);
```

## Description

Set security mechanism for current negotiations

## Parameters

Context	IN
Context to use	
Name	IN
Mechanism name	

## Return Value

TRUE on success, FALSE on failure

## Notes

Since currently NQ uses only one security mechanism, this call may be a dummy. See 6.4.1.12.5 for how to identify a security mechanism.

#### 6.4.1.12.11 sySaslClientGenerateFirstRequest

##### Prototype

```
NQ_BOOL
sySaslClientGenerateFirstRequest(
    NQ_BYTE* context,
    const NQ_CHAR* mechList,
    NQ_BYTE** blob,
    NQ_COUNT* blobLen
);
```

##### Description

Generate first security blob

##### Parameters

context	IN
Context to use	
mechList	IN
List of security mechanisms	
blob	IN
Buffer for a pointer to the first client blob	
blobLen	IN
Buffer for the client blob length	

##### Return Value

TRUE on success, FALSE on failure

##### Notes

NQ calls this function to generate the very first security blob when no response was received yet from the server. SASL, while performing this function may call the user credentials callback (see 6.4.1.12.2).

#### 6.4.1.12.12 sySaslClientGenerateNextRequest

##### Prototype

```
NQ_BOOL
sySaslClientGenerateNextRequest (
    NQ_BYTE* context,
    NQ_BYTE* inBlob,
    NQ_COUNT inBlobLen,
    NQ_BYTE** outBlob,
    NQ_COUNT* outBlobLen
);
```

##### Description

Generate next security blob

##### Parameters

context	IN
Context to use	
inBlob	IN
Pointer to a received server blob	
inBlobLen	IN
Server blob length	
outBlob	OUT
Buffer for a pointer to the next client blob	
outBlobLen	OUT
Buffer for the client blob length	

##### Return Value

TRUE on success, FALSE on failure

##### Notes

NQ calls this function to generate the each next security blob after a response was received from the server. SASL, while performing this function may call the user credentials callback (see 6.4.1.12.2).



## 6.4.1.12.13 sySaslGetSessionKey

## Prototype

```
NQ_BOOL  
sySaslGetSessionKey(  
    NQ_BYTE* context,  
    NQ_BYTE* buffer,  
    NQ_COUNT* len  
);
```

## Description

Withdraw session key

## Parameters

Context	IN
Context to use	
buffer	OUT
Buffer to place session key in	
len	IN
Buffer for the session key length	

## Return Value

TRUE on success, FALSE on failure

## Notes

This call should fill buffer with a 56-128 bit length session key used for data encryption.

#### **6.4.1.13      OpenLDAP Interface**

This group contains definitions, data types and calls for developing applications using OpenLDAP interface. The information in this section is relevant only when UD\_CC\_INCLUDELDAP is defined in *udparams.h*.  
OpenLDAP interface is defined in *syldap.h*.

Please note that this interface is available in Corporate Client only. It requires SASL/GSSAPI/Kerberos interface for proper operation -  
UD\_CC\_INCLUDEEXTENDEDSECURITY and  
UD\_CC\_INCLUDEEXTENDEDSECURITY\_KERBEROS should be defined in *udparams.h*.

Please see NQ Library Reference – LDAP for further reference.

#### 6.4.1.13.1 syLdStart

##### Prototype

```
NQ_STATUS syLdStart(  
    );
```

##### Header file

**syldap.h**

##### Description

Initialize NQ LDAP

##### Return Value

This function returns LDAP\_SUCCESS on success and LDAP\_ERROR on failure.

##### Notes

This function must be called prior to calling other NQ LDAP functions.

## 6.4.1.13.2 syLdStop

## Prototype

```
void syLdStop(  
    );
```

## Header file

**syldap.h**

## Description

Stop NQ LDAP

## Return Value

None

## Notes

This function must be called after application completes using NQ LDAP. This function releases NQ LDAP resources.

### 6.4.1.13.3 syLdConnect

#### Prototype

```
NQ_STATUS syLdConnect(
    const NQ_WCHAR * domain,
    const NQ_WCHAR * dc,
    const NQ_WCHAR * user,
    const NQ_WCHAR * password,
    LDConnectionHandle * handle
);
```

#### Header file

**syldap.h**

#### Description

Connect to LDAP server

#### Parameters

domain	IN	Name of the authenticating domain.
dc	IN	Name of the authenticating domain DC server.
user	IN	Authenticating user.
password	IN	User password.
handle	OUT	Buffer for LDAP connection handle.

#### Return Value

LDAP\_SUCCESS or LDAP\_ERROR

#### Notes

This function attempts to connect LDAP server using provided credentials.

On successful return it fills the `handle` buffer with a connection handle. This value should be used in subsequent NQ LDAP calls over the same connection. This handle should be closed by calling *syLdCloseConnection()*.

The method of locating LDAP server depends on the external LDAP client implementation. The sample implementation of system-abstraction layer assumes that Domain Controller is also the LDAP server.

This function has also ASCII version, accepting string parameters in ASCII.

## 6.4.1.13.4 syLdCloseConnection

## Prototype

```
NQ_STATUS syLdCloseConnection(  
    LDConnectionHandle handle  
);
```

## Header file

**syldap.h**

## Description

Close LDAP connection

## Parameters

handle	IN
LDAP connection handle.	

## Return Value

LDAP\_SUCCESS or LDAP\_ERROR

## Notes

An application should call this function to release a handle allocated in a *syLdConnect()*.  
On success the handle is released, while on failure its status is undefined.

#### 6.4.1.13.5 syLdSearch

##### Prototype

```
NQ_STATUS syLdSearch(
    LDConnectionHandle handle,
    const NQ_CHAR * base,
    NQ_INT scope,
    const NQ_CHAR * filter,
    const NQ_CHAR * attribs[],
    LDResultHandle * result
);
```

##### Header file

**syldap.h**

##### Description

Perform search query

##### Parameters

handle	IN	
		Handle to open connection.
base	IN	
		DN of the base object.
scope	IN	
		Query scope.
filter	IN	
		Search filter. May be NULL.
attribs	IN	
		Pointer to an array of attribute names designating those attributes that should be withdrawn. A NULL value mans a query for all attributes.
result	OUT	
		Buffer for a result handle.

##### Return Value

LDAP\_SUCCESS, LDAP\_ERROR or LDAP\_MOREDATA

##### Notes

This function performs an LDAP search operation. If the operation was succesfull, NQ LDAP returns LDAP\_SUCCESS or LDAP\_MOREDATA and fills a result handle. The application should later release this handle by calling *syLdReleaseResult()*. On any failure this function returns LDAP\_ERROR. In this case the result handle is undefined.

## 6.4.1.13.6 syLdReleaseResult

## Prototype

```
NQ_STATUS syLdReleaseResult(  
    LDConnectionHandle handle,  
    LDResultHandle result  
);
```

## Header file

**syldap.h**

## Description

Release result handle

## Parameters

handle	IN
	Handle to open connection.
result	IN
	Result handle.

## Return Value

LDAP\_SUCCESS or LDAP\_ERROR

## Notes

Application calls this function to release resources allocated for search result.



## 6.4.1.13.7 syLdGetNumEntries

## Prototype

```
NQ_INT syLdGetNumEntries(  
    LDConnectionHandle handle,  
    LDResultHandle result  
);
```

## Header file

**syldap.h**

## Description

Get number of entries in the result

## Parameters

handle	IN
	Handle to open connection.
result	IN
	Result handle.

## Return Value

Number of entries in the result specified by the two handles or LDAP\_ERROR on error

## Notes

This function assumes that both handles specify a valid and active LDAP result. Then it returns number of entries in that result.

## 6.4.1.13.8 syLdGetEntry

## Prototype

```
LDEntryHandle syLdGetEntry(  
    LDConnectionHandle handle,  
    LDResultHandle result,  
    NQ_INT index  
);
```

## Header file

**syldap.h**

## Description

Get result entry by index

## Parameters

Handle	IN
	Handle to open connection.
result	IN
	Result handle.
index	IN
	Entry index.

## Return Value

Entry handle or LDAP\_INVALIDHANDLE on error

## Notes

This function retrieves a handle to the entry specified by its index in the result.

## 6.4.1.13.9 syLdEntryName

## Prototype

```
void syLdEntryName(  
    LDConnectionHandle handle,  
    LDEntryHandle entry,  
    NQ_CHAR * dn  
);
```

## Header file

**syldap.h**

## Description

Get number of entries in the result

## Parameters

handle	IN
	Handle to open connection.
entry	IN
	Entry handle.
dn	OUT
	Buffer for entry's DN.

## Return Value

None

## Notes

This function retrieves name of the entry specified by the two handles. The `entry` value should be retrieved with the `syLdGetEntry ()` function.

## 6.4.1.13.10syLdGetNumAttribs

## Prototype

```
NQ_INT syLdGetNumAttribs(  
    LDConnectionHandle handle,  
    LDEntryHandle entry  
);
```

## Header file

**syldap.h**

## Description

Get number of attributes in an entry

## Parameters

handle	IN
	Handle to open connection.
entry	IN
	Entry handle.

## Return Value

Number of attributes in an entry specified by the two handles or LDAP\_ERROR on error

## Notes

This function assumes that both handles specify a valid and active LDAP object entry. Then it returns number of attributes in that entry.

## 6.4.1.13.11 syLdGetAttrIndexByName

## Prototype

```
NQ_INT syLdGetAttrIndexByName(  
    LDConnectionHandle handle,  
    LDEntryHandle entry,  
    const NQ_CHAR * attr  
);
```

## Header file

**syldap.h**

## Description

Determine attribute index by its name

## Parameters

handle	IN
Handle to open connection.	
entry	IN
Entry handle.	
attr	IN
Attribute name.	

## Return Value

Attribute index in the entry or LDAP\_ERROR

## Notes

This function assumes that both handles specify a valid and active LDAP object entry.

6.4.1.13.12 `syLdGetAttrNamebyIndex`

## Prototype

```
const NQ_CHAR * syLdGetAttrNameByIndex(  
    LDConnectionHandle handle,  
    LDEntryHandle entry,  
    NQ_INT index  
);
```

## Header file

**`syldap.h`**

## Description

Determine attribute name by its index

## Parameters

<code>handle</code>	IN
	Handle to open connection.
<code>entry</code>	IN
	Entry handle.
<code>index</code>	IN
	Attribute index.

## Return Value

Attribute name or NULL

## Notes

This function assumes that both handles specify a valid and active LDAP object entry. An application that called this name should later call `syLdFreeAttrName()` to release memory allocated for this name.

## 6.4.1.13.13 syLdFreeAttrName

## Prototype

```
void syLdFreeAttrName (  
    NQ_CHAR * name  
);
```

## Header file

**syldap.h**

## Description

Free memory allocated for attribute name

## Parameters

name	IN
Name pointer.	

## Return Value

None

## Notes

This function assumes the parameter is a pointer allocated in a *ldGetAttrNamebyIndex()* call.

6.4.1.13.14 `syLdGetNumValues`

## Prototype

```
NQ_COUNT syLdGetNumValues(  
    LDConnectionHandle handle,  
    LDEntryHandle entry,  
    NQ_INT index  
);
```

## Header file

**`syldap.h`**

## Description

Determine number of values for a particular attribute

## Parameters

<code>handle</code>	IN
Handle to open connection.	
<code>entry</code>	IN
Entry handle.	
<code>index</code>	IN
Attribute index.	

## Return Value

Number of values or `LDAP_ERROR`

## Notes

This function assumes that both handles specify a valid and active LDAP object entry and `index` is in the range returned by `syLdGetNumAttribs()`.



#### 6.4.1.13.15 syLdGetAttrValues

##### Prototype

```
const LDValue ** syLdGetAttrValues(
    LDConnectionHandle handle,
    LDEntryHandle entry,
    NQ_INT index
);
```

##### Header file

**syldap.h**

##### Description

Determine number of values for a particular attribute

##### Parameters

handle	IN
	Handle to open connection.
entry	IN
	Entry handle.
index	IN
	Attribute index.

##### Return Value

Double pointer to the array of value descriptors or LDAP\_INVALIDHANDLE on error

##### Notes

This function assumes that both handles specify a valid and active LDAP object entry and `index` is in the range returned by `syLdGetNumAttribs()`. It retrieves an array of value descriptors and returns a pointer to this array. Application can determine the array size by calling `syLdGetNumValues()`. Application is responsible for releasing this array by calling `syLdFreeAttrValues()`.

6.4.1.13.16 `syLdFreeAttrValues`

## Prototype

```
void syLdFreeAttrValues(  
    const LDValue ** handle  
);
```

## Header file

**`syldap.h`**

## Description

Determine number of values for a particular attribute

## Parameters

<code>handle</code>	IN
Handle of attribute values list.	

## Return Value

None

## Notes

This function releases handle allocated in a `syLdGetAttrValues()` call and all memory associated with this handle.

## 6.4.1.13.17syLdDelete

## Prototype

```
NQ_STATUS syLdDelete(  
    LDConnectionHandle handle,  
    const NQ_CHAR * name  
);
```

## Header file

**syldap.h**

## Description

Determine number of values for a particular attribute

## Parameters

handle	IN
	Handle to open connection.
name	IN
	Object DN.

## Return Value

LDAP\_SUCCESS or LDAP\_ERROR

## Notes

This function deletes an object whose DN is specified by name.

## 6.4.1.13.18syLdDeleteAttrib

## Prototype

```
NQ_STATUS syLdDeleteAttrib(  
    LDConnectionHandle handle,  
    const NQ_CHAR * name,  
    const NQ_CHAR * attrib  
);
```

## Header file

**syldap.h**

## Description

Delete object attribute

## Parameters

handle	IN
Handle to open connection.	
name	IN
Object DN.	
attrib	IN
Attribute name.	

## Return Value

LDAP\_SUCCESS or LDAP\_ERROR

## Notes

This function deletes an attribute from an object whose DN is specified by name.

## 6.4.1.13.19syLdAdd

## Prototype

```
LDTransaction syLdAdd(  
    LDConnectionHandle handle,  
    const NQ_CHAR * parent,  
    const NQ_CHAR * name  
    const NQ_CHAR * class  
);
```

## Header file

**syldap.h**

## Description

Start a transaction for add a new object

## Parameters

handle	IN
Handle to open connection.	
parent	IN
Parent object DN.	
name	IN
New object DN.	
class	IN
Object class name.	

## Return Value

Transaction handle or LDAP\_INVALIDHANDLE on error

## Notes

This function creates new object under a specified parent. The newly created object has no attributes. After object creation NQ starts a transaction and return its handle. Later on, application can add information to this transaction. Finally, application must call *ldExecute()* with a TRUE value in its second argument.

## 6.4.1.13.20syLdModify

## Prototype

```
LDTransaction syLdModify(  
    LDConnectionHandle handle,  
    const NQ_CHAR * parent,  
    const NQ_CHAR * name  
);
```

## Header file

**ldapi.h**

## Description

Start a transaction for modifying an existing object

## Parameters

handle	IN
	Handle to open connection.
parent	IN
	Parent object DN.
name	IN
	New object DN.

## Return Value

Transaction handle or LDAP\_INVALIDHANDLE on error

## Notes

This function searches for an object under a specified parent. If this object exists NQ starts a transaction and return its handle. Later on, application can add information to this transaction. Finally, application must call *ldExecute()* with a TRUE value in its second argument.

## 6.4.1.13.21 syLdAddAttributeString

## Prototype

```
NQ_STATUS syLdAddAttributeString(  
    LDTransaction tran,  
    const NQ_CHAR * attr,  
    const NQ_CHAR * value  
);
```

## Header file

**syldap.h**

## Description

Add a string attribute to transaction

## Parameters

tran	IN
Transaction handle.	
attr	IN
Attribute name.	
value	IN
Attribute value.	

## Return Value

NQ\_SUCCEs or NQ\_ERROR

## Notes

This function assumes that an add or modified transaction was open by calling *ldAdd()* or *ldModify()* function. Then it adds a new string attribute to the object specified by this transaction. Information about new attribute is stored locally. The actual attribute will be added to AD after calling *ldExecute()*.

## 6.4.1.13.22syLdAddAttributeBinary

## Prototype

```
NQ_STATUS syLdAddAttributeBinary(  
    LDTransaction tran,  
    const NQ_CHAR * attr,  
    const NQ_BYTE * value,  
    NQ_COUNT len  
);
```

## Header file

**syldap.h**

## Description

Add a binary attribute to transaction

## Parameters

tran	IN
Transaction handle.	
attr	IN
Attribute name.	
value	IN
Attribute value.	
len	IN
Attribute value length in bytes.	

## Return Value

NQ\_SUCCES or NQ\_ERROR

## Notes

This function assumes that an add or modified transaction was open by calling *ldAdd()* or *ldModify()* function .Then it adds a new binary attribute to the object specified by this transaction. Information about new attribute is stored locally. The actual attribute will be added to AD after calling *ldExecute()*.



## 6.4.1.13.23syLdDeleteAttribute

## Prototype

```
NQ_STATUS syLdDeleteAttribute(  
    LDTransaction tran,  
    const NQ_CHAR * attr  
);
```

## Header file

**syldap.h**

## Description

Delete an attribute

## Parameters

Tran	IN
	Transaction handle.
Attr	IN
	Attribute name.

## Return Value

NQ\_SUCCEs or NQ\_ERROR

## Notes

This function assumes that an add or modified transaction was open by *ldModify()* function. Then it deletes an attribute. Attribute is deleted locally. The actual attribute will be deleted in AD after calling *ldExecute()*.

## 6.4.1.13.24syLdExecute

## Prototype

```
NQ_STATUS syLdExecute(  
    LDTransaction tran,  
    NQ_BOOL releaseHandle  
);
```

## Header file

**syldap.h**

## Description

Execute transaction

## Parameters

tran	IN
Transaction handle.	
releaseHandle	IN
TRUE - to finish the transaction, FALSE – to keep it open.	

## Return Value

NQ\_SUCCESS or NQ\_ERROR

## Notes

This function performs a previously prepared transaction. The transaction should be opened with either *ldModify()* or *ldAddAttributeString()*, *ldAddAttributeBinary()* or *ldDeleteAttribute()*.

#### 6.4.2 **Compile Dependent Code**

File SYCOMPIL.H defines parameters, function prototypes, and macro substitutions whose implementation may vary between run-time libraries of different compilers. File SYCOMPIL.C implements functions.

### 6.4.2.1 Compilation Control Directives

This group of preprocessor directives controls how C compiler creates function calls and how it allocates memory for structures. The control of structure packing mechanism involves the following directives:

- *SY\_COMPILERPACK*
- *SY\_PRAGMAPACK\_DEFINED*
- *SY\_PACK\_ATTR*
- *SY\_PACK\_PREFIX*

User should also consider the code in compiler dependent files *sypackon.h* and *sypackof.h*. NQ may use three methods of structure packing:

- If the compiler supports structure attributes, directives *SY\_PACK\_PREFIX* and *SY\_PACK\_ATTR* are used (see below). Then a structure definition looks as:

```
typedef SY_PACK_PREFIX struct {  
    ...  
} SY_PACK_ATTR <name>;
```

In this case directive *SY\_COMPILERPACK* should be defined and directive *SY\_PRAGMAPACK\_DEFINED* should not be defined.

- If the compiler supports structure alignment *pragmas*, a structure definition looks as:

```
#include "sypackon.h"  
typedef SY_PACK_PREFIX struct {  
    ...  
} SY_PACK_ATTR <name>;  
#include "sypackof.h"
```

In this case directives *SY\_PACK\_PREFIX* and *SY\_PACK\_ATTR* should be defined empty, while both *SY\_COMPILERPACK* and *SY\_PRAGMAPACK\_DEFINED* should be defined. Besides, files *sypackon.h* and *sypackof.h* should contain appropriate *#pragma* directives, see below.

- Some compilers do not have any means for structure packing. In this case NQ defines all fields in the appropriate structures as arrays of bytes. This method is suitable for any compiler. However, it reduces CIFS/SMB performance. Therefore, it is not recommended with compilers support one of previous methods.

Additional two files contain compiler dependent code: *sypackon.h* and *sypackof.h*. The code in both of them has effect only when the following parameters are defined: *SY\_COMPILERPACK* and *SY\_PRAGMAPACK\_DEFINED*. User should review those files and choose the most appropriate structure packing directive (usually – a variant of *#pragma ...*).

#### 6.4.2.1.1 SY\_FORCEALLOCATION

##### Prototype

SY\_FORCEALLOCATION

##### Description

Direct the compiler to generate memory allocation code

##### Notes

NQ features two methods of allocating file scope variables:

1. Data storage is dynamically allocated from heap on startup.
2. Data storage is allocated in static variables of a file scope.

When SY\_FORCEALLOCATION is defined the first method is forced.

The first method is more appropriate when NQ NetBios is an application that runs from time to time. In this case the memory allocated by NQ may be later reused.

The second method is preferred when NQ NetBios is always active. With the same memory usage, it grants better target stability by not using heap memory.

#### 6.4.2.1.2 SY\_BIGSTACK

##### Prototype

SY\_BIGSTACK

##### Description

Direct the compiler to generate local buffers on stack

##### Notes

This parameter controls how NQ generates big buffers of a local scope. With this parameter defined NQ assumes that the stack is big enough and it allocates local scope buffers on stack. When this parameter is not defined NQ allocates local buffers as static variables of the local scope.

Defining this parameter is more appropriate when NQ is an application that runs from time to time. In this case the memory allocated by NQ may be later reused.

Masking this parameter is preferred when NQ is always active. With the same memory usage, it requires less stack size.

#### 6.4.2.1.3 SY\_DEBUGMODE

Prototype

SY\_DEBUGMODE

Description

The way to determine that NQ runs in debug mode

Notes

This compile-time value defines whether NQ runs in debug mode.

#### 6.4.2.1.4 SY\_STARTAPI

##### Prototype

`SY_STARTAPI`

##### Description

Starts a block of ultimate C call convention

##### Notes

NQ uses this code in API definition files. It applies to cases when those files can be included into C++ source.

This call should is likely to be implemented as a preprocessor substitution. For C++ compilation it should specify C convention, while for C compiler this substitution should be empty.



## 6.4.2.1.5   SY\_ENDAPI

## Prototype

`SY_ENDAPI`

## Description

Closes a block of ultimate C call convention

## Notes

NQ uses this code in API definition files. It applies to cases when those files can be included into C++ source.

This call should is likely to be implemented as a preprocessor substitution. For C++ compilation it should specify C convention, while for C compiler this substitution should be empty.

#### 6.4.2.1.6 SY\_COMPILERPACK

##### Prototype

`SY_COMPILERPACK`

##### Description

Defines whether the compiler can pack structures

##### Notes

When defined this directive signals that the compiler has some mean of packing structures. The actual structure packing mechanism is defined by `SY_PRAGMAPACK_DEFINED` or `SY_PACK_PREFIX` or `SY_PACK_ATTR` directives.

When `SY_COMPILERPACK` is not defined `SY_PRAGMAPACK_DEFINED` should be also undefined while `SY_PACK_PREFIX` and `SY_PACK_ATTR` should be defined an empty strings.

#### 6.4.2.1.7 SY\_PRAGMAPACK\_DEFINED

##### Prototype

`SY_PRAGMAPACK_DEFINED`

##### Description

Direct the compiler to pack structures

##### Notes

This substitution should directive should be the only directive in a line.

This substitution uses compiler-specific directives to turn structure packing off and on again.

The `SY_PRAGMAPACK_DEFINED` compile-time parameter is complimentary to `SY_PACK_PREFIX` and `SY_PACK_ATTR` directives. Either the first two or the last one should be empty.

This directive should not be defined when `SY_COMPILERPACK` is not defined.

#### 6.4.2.1.8 SY\_PACK\_PREFIX

##### Prototype

```
struct SY_PACK_PREFIX {  
    ...  
} ... ;
```

##### Description

Direct the compiler to pack the current structure

##### Notes

This substitution should directive should be the only directive in a line.

This substitution uses compiler-specific directives to turn structure packing off. Members of a structure with this attribute will be aligned in memory.

The SY\_PRAGMAPACK\_DEFINED compile-time parameter is complimentary to SY\_PACK\_PREFIX directive. Either the first two or the last one should be empty.

When SY\_COMPILERPACK is not defined, SY\_PACK\_ATTR should be empty.

This directive is optional. When not defined explicitly it will be evaluated to an empty string

#### 6.4.2.1.9 SY\_PACK\_ATTR

##### Prototype

```
struct ... {  
    ...  
} SY_PACK_ATTR ... ;
```

##### Description

Direct the compiler to pack the current structure

##### Notes

This substitution directive should be the only directive in a line.

This substitution uses compiler-specific directives to turn structure packing off. Members of a structure with this attribute will be aligned in memory.

The SY\_PRAGMAPACK\_DEFINED compile-time parameter is complimentary to SY\_PACK\_ATTR directive. Either the first two or the last one should be empty.

When SY\_COMPILERPACK is not defined, SY\_PACK\_ATTR should be empty.

This directive is optional. When not defined explicitly it will be evaluated to an empty string

## 6.4.2.1.10 SY\_INT32

## Prototype

`SY_INT32`

## Description

Define a 32bit type

## Notes

This definition is optional. NQ needs to know which data type is exactly 32bit long. By default, the “long” type is used. Therefore, for those compilers that have 4-byte longs, this definition should be omitted. If the above is not true define this macro as a 4-byte type. For instance, if “int” is a 4-byte value on your target architecture, define the following:

```
#define SY_INT32 int
```

This parameter should be defined as a primitive type only without any qualifiers like “unsigned”.

## 6.4.2.1.11 SY\_C99\_MACRO

## Prototype

`SY_C99_MACRO`

## Description

Enable C99 macros

## Notes

This definition is optional. When this parameter is not defined, NQ code does not use C99-specific macros (e.g., - \_\_VA\_ARGS\_\_). By default this parameter is omitted.

Define this parameter to force usage of several C99-specific macros. This option is necessary when the target compiler requires C99 syntax, for instance – in variable argument macros.

#### **6.4.2.2 Byte Order Conversion**

NQ uses the functions in this group to convert 16 and 32-bit values from host to network byte order and vice versa.



#### 6.4.2.2.1 syNtoh32

## Prototype

```

NQ_UINT32 syNtoh32(
    NQ_UINT32 nbo
);

```

### Description

## Convert 32-bit value from NBO to HBO

## Parameters

nbo	IN
Source value in NBO	

### Return Value

Source value converted to HBO

## Notes

This call is time critical and it should use the quickest conversion from Network Byte Order (NBO) to Host Byte Order (HBO). NQ software takes into consideration that the system implementation of this function may be a macro.

## 6.4.2.2.2 syNtoh16

## Prototype

```
NQ_UINT16 syNtoh16(  
    NQ_UINT16 nbo  
);
```

## Description

Convert 16-bit value from NBO to HBO

## Parameters

nbo	IN
Source value in NBO	

## Return Value

Source value converted to HBO

## Notes

This call is time critical and it should use the quickest conversion from Network Byte Order (NBO) to Host Byte Order (HBO). NQ software takes into consideration that the system implementation of this function may be a macro.

#### 6.4.2.2.3 syHton32

## Prototype

```

NQ_UINT32 syHton32(
    NQ_UINT32 hbo
);

```

### Description

## Convert 32-bit value from HBO to NBO

## Parameters

hbo	IN
Source value in HBO	

### Return Value

Source value converted to NBO

## Notes

This call is time critical and it should use the quickest conversion from Host Byte Order (HBO) to Network Byte Order (NBO). NQ software takes into consideration that the system implementation of this function may be a macro.

#### 6.4.2.2.4 syHton16

## Prototype

```

NQ_UINT16 syHton16(
    NQ_UINT16 hbo
);

```

### Description

## Convert 16-bit value from HBO to NBO

## Parameters

hbo	IN
Source value in HBO	

### Return Value

Source value converted to NBO

## Notes

This call is time critical and it should use the quickest conversion from Host Byte Order (HBO) to Network Byte Order (NBO). NQ software takes into consideration that the system implementation of this function may be a macro.

#### **6.4.2.3 String and Memory Manipulation**

This group contains analogues of string and memory manipulation usually found in <STRING.H>.

## 6.4.2.3.1 syMemcpy

## Prototype

```
void *syMemcpy(  
    void *dst,  
    const void *source,  
    NQ_COUNT size  
);
```

## Description

Copy memory from one location to another

## Parameters

destination	OUT
Destination buffer	
source	IN
Pointer to the sources data	
size	IN
Number of bytes to copy	

## Return Value

A pointer to *destination*

## Notes

This function is an analog of POSIX *memcpy* function.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

## 6.4.2.3.2 syMemmove

## Prototype

```
void *syMemmove(  
    void *dst,  
    const void *source,  
    NQ_COUNT size  
);
```

## Description

Copy memory from one location to another preventing overlapping

## Parameters

destination	OUT
Destination buffer	
source	IN
Pointer to the sources data	
size	IN
Number of bytes to copy	

## Return Value

A pointer to *destination*

## Notes

This function is an analog of POSIX *memmove* function.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

## 6.4.2.3.3 syMemset

## Prototype

```
void *syMemset(  
    void *block,  
    NQ_INT sample,  
    NQ_COUNT size  
);
```

## Description

Fill a block of memory

## Parameters

<code>block</code>	OUT
Address of the block of memory to fill	
<code>sample</code>	IN
Use this value for filling the block	
<code>size</code>	IN
Number of bytes to fill with <i>sample</i>	

## Return Value

A pointer to *block*

## Notes

This function is an analog of POSIX *memset* function.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.



## 6.4.2.3.4 syMemcmp

## Prototype

```
NQ_INT syMemcmp(  
    const void *p1,  
    const void *p2,  
    NQ_COUNT size  
);
```

## Description

Compare two blocks of memory until unequal elements are found

## Parameters

p1	IN
	Address of the first memory block
p2	IN
	Address of the second memory block
size	IN
	Number of bytes to compare

## Return Value

0 if all *size* elements of both arrays are equal, positive number if the different element from *p1* is greater than the element from *p2*, negative number if the different element from *p1* is less than the element from *p2*.

## Notes

This function is an analog of POSIX *memcmp* function.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

## 6.4.2.3.5 syStrlen

## Prototype

```
NQ_COUNT syStrlen(  
    const NQ_CHAR *str  
);
```

## Description

Find number of characters in the string

## Parameters

<code>str</code>	IN
Pointer to the string of interest	

## Return Value

Number of characters in the string not including zero terminator

## Notes

This function is an analog of POSIX *strlen* function.  
Since this call is time critical it is likely to be implemented as a  
macro mapping it on the system-dependent function.

## 6.4.2.3.6 syStrcpy

## Prototype

```
NQ_CHAR *syStrcpy(  
    NQ_CHAR *destination,  
    const NQ_CHAR *source  
);
```

## Description

Copy one string into another

## Parameters

destination	OUT
	Pointer to the destination string
source	IN
	Pointer to the source string

## Return Value

Pointer to *destination*

## Notes

This function is an analog of POSIX *strcpy* function.  
Since this call is time critical it is likely to be implemented as a  
macro mapping it on the system-dependent function.

## 6.4.2.3.7 syStrncpy

## Prototype

```
NQ_CHAR *syStrncpy(  
    NQ_CHAR *destination,  
    const NQ_CHAR *source,  
    NQ_COUNT size  
);
```

## Description

Copy number of bytes from one string into another

## Parameters

destination	OUT
	Pointer to the destination string
source	IN
	Pointer to the source string
size	IN
	Number of characters to copy

## Return Value

Pointer to *destination*

## Notes

This function is an analog of POSIX *strcpy* function.  
Since this call is time critical it is likely to be implemented as a  
macro mapping it on the system-dependent function.

## 6.4.2.3.8 syStrcmp

## Prototype

```
NQ_INT syStrcmp(  
    const NQ_CHAR *s1,  
    const NQ_CHAR *s2  
);
```

## Description

Compare two strings

## Parameters

s1	IN
	Pointer to the first string
s2	IN
	Pointer to the second string

## Return Value

0 if both strings are equal, positive number if the first non-matched character from *s1* is greater than the element from *s2*, negative number if the first non-matched character from *s1* is less than the element from *s2*.

## Notes

This function is an analog of POSIX *strcmp* function. Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

## 6.4.2.3.9 syStrncmp

## Prototype

```
NQ_INT syStrncmp(  
    const NQ_CHAR *s1,  
    const NQ_CHAR *s2,  
    NQ_COUNT max  
);
```

## Description

Compare beginning of two strings

## Parameters

s1	IN
	Pointer to the first string
s2	IN
	Pointer to the second string
max	IN
	Maximum number of characters to compare

## Return Value

0 if both strings are equal or the first *max* characters of both strings are equal, positive number if the first non-matched character from *s1* is greater than the element from *s2*, negative number if the first non-matched character from *s1* is less than the element from *s2*.

## Notes

This function is an analog of POSIX *strncmp* function.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

## 6.4.2.3.10 syStrcat

## Prototype

```
NQ_CHAR *syStrcat(  
    NQ_CHAR *destination,  
    const NQ_CHAR *append  
);
```

## Description

Concatenate one string to another

## Parameters

destination	OUT
	Pointer to the destination string
append	IN
	Pointer to the string to concatenate with <i>destination</i>

## Return Value

Pointer to *destination*

## Notes

This function is an analog of POSIX *strcat* function.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

## 6.4.2.3.11 syStrncat

## Prototype

```
NQ_CHAR *syStrncat(  
    NQ_CHAR *destination,  
    const NQ_CHAR *append,  
    NQ_COUNT number  
);
```

## Description

Concatenate up to *number* characters from one string to another

## Parameters

<i>destination</i>	OUT
Pointer to the destination string	
<i>append</i>	IN
Pointer to the string to concatenate with <i>destination</i>	
<i>number</i>	IN
Maximum number of characters to append	

## Return Value

Pointer to *destination*

## Notes

This function is an analog of POSIX *strncat* function.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.



## 6.4.2.3.12 syToupper

## Prototype

```
NQ_CHAR syToupper(  
    NQ_CHAR source  
);
```

## Description

Convert one character to uppercase

## Parameters

`source`                                      IN  
Character to convert

## Return Value

Uppercase character corresponding to *source*

## Notes

This function is an analog of POSIX *toupper* call.  
Since this call is time critical it is likely to be implemented as a  
macro mapping it on the system-dependent function.  
NQ software takes into consideration that the system implementation  
of this function may be a macro.

## 6.4.2.3.13 syStrchr

## Prototype

```
NQ_CHAR *syStrchr(  
    const NQ_CHAR *string,  
    NQ_CHAR ch  
);
```

## Description

Find the first occurrence of a character in a string

## Parameters

<code>string</code>	IN
	Pointer to the string of the interest
<code>ch</code>	IN
	Character to find

## Return Value

Pointer to the first occurrence of *ch* inside *string* or NULL if the character was not found

## Notes

This function is an analog of POSIX *strchr* call.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

## 6.4.2.3.14 syStrchr

## Prototype

```
NQ_CHAR *syStrchr(  
    const NQ_CHAR *string,  
    NQ_CHAR ch  
);
```

## Description

Find the last occurrence of a character in a string

## Parameters

<code>string</code>	IN
	Pointer to the string of the interest
<code>ch</code>	IN
	Character to find

## Return Value

Pointer to the last occurrence of *ch* inside *string* or NULL if the character was not found

## Notes

This function is an analog of POSIX *strrchr* call.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

## 6.4.2.3.15 syStrstr

## Prototype

```
NQ_CHAR * syStrstr(  
    const NQ_CHAR *str1,  
    const NQ_CHAR *str2  
);
```

## Description

Compare two strings

## Parameters

string	IN
	Pointer to the string to be scanned
string	IN
	String containing the sequence of characters to match

## Return Value

A pointer to the first occurrence in str1 of the entire sequence of characters specified in str2, or a null pointer if the sequence is not present in str1.

## Notes

This function is an analog of POSIX *strstr* function.  
Since this call is time critical it is likely to be implemented as a macro mapping it on the system-dependent function.

#### 6.4.2.4 Unicode String Manipulation

This group contains functions for manipulation with two-byte Unicode strings.

NQ software contains a set of platform-independent functions for this purpose. This allows the implementation of this group in SYCOMPIL.H to be a substitution for NQ functions, as follows from the following example:

```
#define syWStrlen  cmWStrlen
```

Where:

*syWStrlen*    is a prototype for platform-dependent function

*cmWStrlen*    is NQ implementation for this function

However, the target platform may feature quicker functions for the same purpose. In this case the functions of this group may be defined by means of respective system functions.

## 6.4.2.4.1 syWStrlen

## Prototype

```
NQ_UINT syWStrlen(  
    const NQ_WCHAR *str  
);
```

## Description

Find number of characters in the Unicode string

## Parameters

<code>str</code>	IN
Pointer to the string of interest	

## Return Value

Number of two-byte characters in the string not including zero terminator

## Notes

This function may be mapped on NQ *cmWStrlen* function.

## 6.4.2.4.2 syWStrcpy

## Prototype

```
NQ_WCHAR *syWStrcpy(  
    NQ_WCHAR *destination,  
    const NQ_WCHAR *source  
);
```

## Description

Copy one Unicode string into another

## Parameters

destination	OUT
	Pointer to the destination string
source	IN
	Pointer to the source string

## Return Value

Pointer to *destination*

## Notes

This function may be mapped on NQ *cmWStrcpy* function.

#### 6.4.2.4.3 syWStrncpy

##### Prototype

```
NQ_WCHAR *syWStrncpy(
    NQ_WCHAR *destination,
    const NQ_WCHAR *source,
    NQ_COUNT size
);
```

##### Description

Copy number of bytes from one Unicode string into another

##### Parameters

destination	OUT
	Pointer to the destination string
source	IN
	Pointer to the source string
size	IN
	Buffer size – the maximum number of two-byte characters that may be copied including the terminating zero

##### Return Value

Pointer to *destination*

##### Notes

This function may be mapped on NQ *cmWStrncpy* function.



## 6.4.2.4.4 syWStrcmp

## Prototype

```
NQ_INT syWStrcmp(  
    const NQ_WCHAR *s1,  
    const NQ_WCHAR *s2  
);
```

## Description

Compare two Unicode strings

## Parameters

s1	IN
	Pointer to the first string
s2	IN
	Pointer to the second string

## Return Value

0 if both strings are equal, positive number if the first non-matched character from *s1* is greater than the element from *s2*, negative number if the first non-matched character from *s1* is less than the element from *s2*.

## Notes

This function may be mapped on NQ *cmWStrcmp* function.

#### 6.4.2.4.5 syWStrncmp

##### Prototype

```
NQ_INT syWStrncmp(  
    const NQ_WCHAR *s1,  
    const NQ_WCHAR *s2,  
    NQ_COUNT max  
);
```

##### Description

Compare beginning of two Unicode strings

##### Parameters

s1	IN	
		Pointer to the first string
s2	IN	
		Pointer to the second string
max	IN	
		Maximum number of two-byte characters to compare

##### Return Value

0 if both strings are equal or the first *max* characters of both strings are equal, positive number if the first non-matched character from *s1* is greater than the element from *s2*, negative number if the first non-matched character from *s1* is less than the element from *s2*.

##### Notes

This function may be mapped on NQ *cmWStrcmp* function.

## 6.4.2.4.6 syWStricmp

## Prototype

```
int syWStricmp(  
    const NQ_WCHAR *s1,  
    const NQ_WCHAR *s2  
);
```

## Description

Compare two Unicode strings ignoring case

## Parameters

s1	IN
	Pointer to the first string
s2	IN
	Pointer to the second string

## Return Value

0 if both strings are equal or the first *max* characters of both strings are equal, positive number if the first non-matched character from *s1* is greater than the element from *s2*, negative number if the first non-matched character from *s1* is less than the element from *s2*.

## Notes

This function may be mapped on NQ *cmWStricmp* function.

## 6.4.2.4.7 syWStrcat

## Prototype

```
NQ_WCHAR *syWStrcat(  
    NQ_WCHAR *destination,  
    const NQ_WCHAR *append  
);
```

## Description

Concatenate one Unicode string to another

## Parameters

destination	OUT
	Pointer to the destination string
append	IN
	Pointer to the string to concatenate with <i>destination</i>

## Return Value

Pointer to *destination*

## Notes

This function may be mapped on NQ *cmWStrcat* function.

## 6.4.2.4.8 syWStrchr

## Prototype

```
NQ_WCHAR *syWStrchr(  
    const NQ_WCHAR *string,  
    NQ_WCHAR ch  
);
```

## Description

Find the first occurrence of a Unicode character in a Unicode string

## Parameters

<code>string</code>	IN
	Pointer to the string of the interest
<code>ch</code>	IN
	Two-byte character to find

## Return Value

Pointer to the first occurrence of *ch* inside *string* or NULL if the character was not found

## Notes

This function may be mapped on NQ *cmWStrchr* function.

## 6.4.2.4.9 syWStrrchr

## Prototype

```
NQ_WCHAR *syWStrrchr(  
    const NQ_WCHAR *string,  
    NQ_WCHAR ch  
);
```

## Description

Find the last occurrence of a Unicode character in a Unicode string

## Parameters

<code>string</code>	IN
	Pointer to the string of the interest
<code>ch</code>	IN
	Two-byte character to find

## Return Value

Pointer to the last occurrence of *ch* inside *string* or NULL if the character was not found

## Notes

This function may be mapped on NQ *cmWStrrchr* function.

#### 6.4.2.4.10 syUnicodeToUTF8N

Prototype			
	<pre>void syUnicodeToUTF8N(     NQ_CHAR *outStr,     NQ_UINT outLength,     const NQ_WCHAR *inWStr,     NQ_UINT inLength );</pre>		
Description			
	Convert Unicode UTF-16LE string to UTF8		
Parameters			
	outStr	OUT	
		Pointer to the result string in UTF8	
	outLength	IN	
		Length of the result buffer	
	inWStr	IN	
		Pointer to string in UNICODE to be converted	
	inLength	IN	
		Length of string to be converted	
Return Value			
	None		
Notes			

#### 6.4.2.4.11 syUTF8ToUnicodeN

Prototype			
	<pre>void syUTF8ToUnicodeN(     NQ_WCHAR *outWstr,     NQ_UINT outLength,     const NQ_CHAR *inStr,     NQ_COUNT inLength );</pre>		
Description			
	Convert Unicode UTF-16LE string to UTF8		
Parameters			
	outWstr	OUT	
		Pointer to the result string in UNICODE	
	outLength	IN	
		Result buffer length	
	inStr	IN	
		Pointer to input string in UTF8	
	inLength	IN	
		Length of input string	
Return Value			
	None		
Notes			



#### **6.4.2.5 System Errors**

Some system calls that NQ issues do not return a distinct error code, but rather an indication of the fact that either error happened. NQ requires a mechanism of reading and writing system error code. This mechanism is implemented by functions of this group.

## 6.4.2.5.1 syGetLastError

## Prototype

```
NQ_INT syGetLastError(  
    void  
);
```

## Description

Find the code of the last system error

## Parameters

None

## Return Value

Error code

## Notes

The POSIX analog of this function is *errno*. This call should never return a zero value after an error has happened. For instance, the *syOpenFileForRead()* can call the *open()* function of the underlying file system. If the call to *open()* fails, a subsequent call to *syGetLastError()* should return a non-zero value. In a POSIX-compliant environment this is always the case since *errno* carries a non-zero error code.

## 6.4.2.5.2 sySetLastError

## Prototype

```
void sySetLastError(  
    NQ_INT err  
);
```

## Description

Set system error code

## Parameters

<code>err</code>	IN
	Error code

## Return Value

None

## Notes

The POSIX analog is `errno = err;`

#### **6.4.2.6 Placeholder File**

NQ provides a placeholder file SYCOMPIL.C. All compiler dependent functions should be implemented here.

This file implements one function that is also a placeholder. It is never referenced in the NQ code; therefore there is no reason to modify its implementation. The only reason for implementing this function is avoiding compilation problems on empty files. This function may be safely removed. This, however, may cause compilation-time warning in definite compilers.

## 6.4.2.6.1 syCompilerInit

## Prototype

```
void syCompilerInit(  
    void  
);
```

## Description

Placeholder

## Parameters

None

## Return Value

None

## Notes

This function may be safely removed. This, however, may cause compilation-time warning in some compilers.

### **6.4.3 Hardware Dependent Code**

Functions in this group may be implemented differently when porting NQ from one target hardware (board) to another, while the same operating system and the compiler remain unchanged.

NQ provides a placeholder file SYPLATFORM.C. All compiler dependent functions should be implemented here.

This file implements one function that is also a placeholder - syPlatformInit. It is never referenced in the NQ code; therefore there is no reason to modify its implementation. The only reason for implementing this function is avoiding compilation problems on empty files. This function may be safely removed. This, however, may cause compilation-time warning in definite compilers.

### 6.4.3.1 syGetTimeAccuracy

Prototype

```
NQ_UINT syGetTimeAccuracy(  
    void  
);
```

Description

Get the accuracy of the system timer

Parameters

None

Return Value

Timer precision in 100 nanosecond units

Notes

This call returns the minimum time interval that this hardware may measure (clock tick).

This call may be substituted by a hardware-defined constant

### 6.4.3.2 SY\_LITTLEENDIANHOST

Prototype

`SY_LITTLEENDIANHOST`

Description

Define this target as Little Endian

Notes

If this pre-compile directive is defined, SY\_BIGENDIANHOST (6.4.3.3) should not be defined



### 6.4.3.3 SY\_BIGENDIANHOST

Prototype

SY\_BIGENDIANHOST

Description

Define this target as Big Endian

Notes

If this pre-compile directive is defined, SY\_LITTLEENDIANHOST (6.4.3.2) should not be defined

## 6.4.3.3.1 syPlatformInit

## Prototype

```
void syPlatformInit(  
    void  
);
```

## Description

Placeholder

## Parameters

None

## Return Value

None

## Notes

This function may be safely removed. This, however, may cause compilation-time warning in some compilers.

#### **6.4.4 Trace System**

Functions in this group are used in debug mode (see 6.4.2.1.1) to produce trace printout.

This group contains one file – SYTRACE.H. For most Operating Systems the calls in this file are implemented as macros.

#### 6.4.4.1 LOGMSG

##### Prototype

```
void LOGMSG (
    NQ_UINT level,
    const NQ_CHAR *text,
    ...
);
```

##### Description

Print text message

##### Parameters

level	IN
	Log message threshold
Text	IN
<i>Optional</i>	Message text
...	IN
<i>Optional</i>	In parameters

##### Return Value

None

##### Notes

Typical implementation will print the following:

- File name
- Function name
- Line number in the source code
- Text message as specified in *text*
- New line character

### 6.4.4.2 LOGERR

#### Prototype

```
void LOGERR(
    NQ_UINT level,
    const NQ_CHAR *text,
    ...
);
```

#### Description

Print error message

#### Parameters

<code>level</code>	IN
	Log message threshold
<code>Text</code>	IN
<i>Optional</i>	Message text
<code>...</code>	IN
<i>Optional</i>	In parameters

#### Return Value

None

#### Notes

Typical implementation will print the following:

- File name
- Function name
- Line number in the source code
- Text message as specified in *text*
- New line character

### 6.4.4.3 LOGFB

#### Prototype

```
void LOGFB(
    NQ_UINT level,
    const NQ_CHAR *text,
    ...
);
```

#### Description

Use in the start of the function

#### Parameters

<code>level</code>	IN
	Log message threshold
<code>Text</code>	IN
<i>Optional</i>	Message text
<code>...</code>	IN
<i>Optional</i>	In parameters

#### Return Value

None

#### Notes

Typical implementation will print the following:

- File name
- Function name
- Line number in the source code
- Text message as specified in *text*
- New line character

#### 6.4.4.4 LOGFE

##### Prototype

```
void LOGFB(
    NQ_UINT level,
    const NQ_CHAR *text,
    ...
);
```

##### Description

Use in the exit of the function

##### Parameters

<code>level</code>	IN
	Log message threshold
<code>Text</code>	IN
<i>Optional</i>	Message text
<code>...</code>	IN
<i>Optional</i>	In parameters

##### Return Value

None

##### Notes

Typical implementation will print the following:

- File name
- Function name
- Line number in the source code
- Text message as specified in *text*
- New line character

### 6.4.4.5 LOGDUMP

#### Prototype

```
void LOGFB(
    NQ_UINT level,
    const NQ_CHAR *text,
    const void *bufferAddr,
    NQ_UINT bufferSize
);
```

#### Description

Use in the start of the function

#### Parameters

level	IN
	Log message threshold
Text	IN
	Message text
bufferAddr	IN
	The pointer to the buffer to “dump”
BufferSize	IN
	The buffer size

#### Return Value

None

#### Notes

Typical implementation will print the following:

- File name
- Function name
- Line number in the source code
- Text message as specified in *text*
- New line character



#### 6.4.4.6 syTraceInit

Prototype

```
void syTraceInit();
```

Description

External tracer initialization. NQ issues this call once on its startup

Parameters

None

Return Value

None

Notes

This function is for external implementation only.

#### 6.4.4.7 syTraceShutdown

Prototype

```
void syTraceShutdown();
```

Description

External tracer shutdown. NQ issues this call once on its shutdown

Parameters

None

Return Value

None

Notes

This function is for external implementation only.

#### 6.4.4.8 syTraceMessage

##### Prototype

```
void syTraceMessage(
    const NQ_CHAR *file,
    const NQ_CHAR *function,
    NQ_UINT line,
    NQ_UINT level,
    NQ_UINT type,
    const NQ_CHAR *format,
    ...
);
```

##### Description

Create a trace message. This call may contain a variable argument list in accordance with the `_format` parameter

##### Parameters

<code>_file</code>	IN	Name of the source file
<code>_function</code>	IN	Name of the enclosing function
<code>_line</code>	IN	Line number in the source file
<code>_level</code>	IN	Trace priority
<code>_type</code>	IN	Record type as: 0 - function call entrance. 1 - function return. 2 - error message. 3 - generic text message. 4 - task/thread start. 5 - task/thread shutdown.
<code>_format</code>	IN	The format string in the same form as used in a printf call

##### Return Value

None

##### Notes

This function is for external implementation only.

#### 6.4.4.9 syTraceDump

##### Prototype

```
void syTraceDump (
    _file,
    _function,
    _line,
    _level,
    _str,
    _addr,
    _nBytes );
```

##### Description

Dump an array of bytes into Trace Log

##### Parameters

<code>_file</code>	IN	Name of the source file
<code>_function</code>	IN	Name of the enclosing function
<code>_line</code>	IN	Line number in the source file
<code>_level</code>	IN	Trace priority
<code>_str</code>	IN	Explanatory text
<code>_addr</code>	IN	Pointer to the first byte to dump
<code>_nBytes</code>	IN	Number of bytes to dump

##### Return Value

None

##### Notes

This function is for external implementation only.

#### 6.4.4.10 syTraceIODump

##### Prototype

```
void syTraceIODump(
    _file,
    _function,
    _line,
    _level,
    _str,
    _addr,
    _nBytes );
```

##### Description

Dump an array of bytes into Trace Log

##### Parameters

<code>_file</code>	IN	Name of the source file
<code>_function</code>	IN	Name of the enclosing function
<code>_line</code>	IN	Line number in the source file
<code>_level</code>	IN	Trace priority
<code>_str</code>	IN	Explanatory text
<code>_addr</code>	IN	Pointer to the first byte to dump
<code>_nBytes</code>	IN	Number of bytes to dump

##### Return Value

None

##### Notes

This function is for external implementation only.

## 7 Integration Guide

### 7.1 Integration Process

Integration process stands for making those modifications that are specific for a given project. This process concerns one module – UD. The code in this module is divided into two parts: run-time functions that may be rewritten or modified and compile time parameters that defined the behavior of entire NQ software. Binary code license is enough for modifying run-time functions, while changing compilation parameters requires sources code license.

### 7.2 Run-time Functions

These prototypes reside in file UDAPI.H. NQ provides user with default implementation. This implementation consists of the following files:

UDAPI.C	This file contains placeholders for functions of the run-time parameters group (7.2). Each function in this file calls a respective default function from UDCONFIG.C. It is recommended to place project-dependent code in place of default function call.
UDCONFIG.H	This file contains prototypes for default implementation of functions of the run-time parameters group (7.2). Each default function has exactly the same name as a respective API function except for <i>Def</i> addition. For example: function <i>udDefGetDomain</i> is a default implementation for <i>udGetDomain</i> .
UDCONFIG.C	This file contains default implementations of the functions from the run-time parameter group (7.2). It is not recommended to modify this file. A user is expected to place project-dependent implementations in UDAPI.H.
UDDESCRP.C	This file contains default implementations for the functions from the security descriptors group (7.2.2.18).
UDERRORS.C	This file contains default implementations for the functions from the error code conversions group (7.2.5).
UDPARAMS.H	This file contains main NQ configuration parameters.
UDPARSER.H	This file contains function prototypes and data types for parsing configuration files. This file applies to the default implementation and is not mandatory for a project-dependent implementation.
UDPARSER.C	This file contains code for parsing configuration files. This file applies to the default implementation and is not mandatory for a project-dependent implementation.

### 7.2.1 Synchronization Calls

NQ software may be synchronized with the project software so that project software will be aware of certain events inside NQ. For this purpose NQ uses a set of functions – synchronization calls. Each of functions is called on certain event inside NQ. The sample implementation of these functions is empty.

Since synchronizations calls are executed in the context of NQ tasks, the recommended implementation is mere signaling to other tasks.

### 7.2.1.1 udNetBiosDaemonStarted

Prototype

```
void udNetBiosDaemonStarted(  
    void  
);
```

Description

Signal to the user level that NetBIOS Daemon has started up.

Parameters

None

Return Value

None

Notes

This function is called few seconds after ndStart() call was issued (see 5.4.1). The sample implementation of this function is a placeholder. A user may use this function to synchronize between NetBIOS Daemon and the application software.



### 7.2.1.2 udNetBiosDaemonClosed

Prototype

```
void udNetBiosDaemonClosed(  
    void  
);
```

Description

Signal to the user level that NetBIOS Daemon was shut down

Parameters

None

Return Value

None

Notes

This function is called as a result of `ndStop()` call (see 5.4.2). When this function was called, NetBIOS Daemon is fully closed and does not accept connections.

The sample implementation of this function is a placeholder. A user may use this function to synchronize between NetBIOS Daemon and the application software.

### 7.2.1.3 udNetBiosDataIn

Prototype

```
void udNetBiosDataIn(  
    void  
);
```

Description

Project-level processing on an incoming message for NQ NetBIOS Daemon

Parameters

None

Return Value

None

Notes

NQ NetBIOS Daemon calls this function when incoming data is detected on one of its sockets.  
The implementation of this function may include any project-level processing

#### 7.2.1.4 udNetBiosError

##### Prototype

```
void udNetBiosError(
    NQ_STATUS cause,
    const NQ_CHAR* name
);
```

##### Description

A callback function to inform the project level of a NetBIOS error

##### Parameters

cause	IN
	Error code
name	IN
	NetBIOS name that caused the error

##### Return Value

None

##### Notes

The *cause* parameter can contain one of the following values:

CM\_NBERR\_NOTNETBIOSNAME  
 CM\_NBERR\_TIMEOUT  
 CM\_NBERR\_NEGATIVERESPONSE  
 CM\_NBERR\_HOSTNAMENOTRESOLVED  
 CM\_NBERR\_CANCELLISTENFAIL  
 CN\_NBERR\_SOCKETOVERFLOW  
 CM\_NBERR\_NOBINDBEFORELISTEN  
 CM\_NBERR\_ILLEGALDATAGRAMSOURCE  
 CM\_NBERR\_ILLEGALDATAGRAMDESTINATION  
 CM\_NBERR\_INVALIDPARAMETER  
 CM\_NBERR\_INTERNALERROR  
 CM\_NBERR\_ILLEGALDATAGRAMTYPE  
 CM\_NBERR\_DDCOMMUNICATIONERROR  
 CM\_NBERR\_BUFFEROVERFLOW  
 CM\_NBERR\_RELEASENAMEFAIL

Right now only CM\_NBERR\_NEGATIVERESPONSE is used while others are reserved for a future use.

### 7.2.2 Run-Time Parameters

Functions in this group provide NQ with project dependent parameters that may be determined only when NQ is executed. Function *udInit()* is called first. Its aim is to prepare for subsequent functions.

### 7.2.2.1 udInit

Prototype

```
NQ_STATUS udInit(  
    void  
);
```

Description

Start the UD module

Parameters

None

Return Value

NQ\_SUCCESS or NQ\_FAIL

Notes

NQ calls this function on its start up. Its aim is to initialize resources, provided by subsequent calls. Right now NQ does not consider the returned value.

### 7.2.2.2 udStop

Prototype

```
void udStop(  
    void  
);
```

Description

Stop the UD module

Parameters

None

Return Value

None

Notes

UD module may use this function for releasing resources.

### 7.2.2.3 udGetScopeID

Prototype

```
void udGetScopeID(  
    NQ_WCHAR *buffer  
);
```

Description

Read scope ID string from project-specific resource

Parameters

buffer	OUT
Pointer to buffer for scope ID. This buffer's length is UD_NS_SCOPEIDLEN bytes (see 7.3.1) including terminating zero.	

Return Value

None

Notes

NQ calls this function once during the start up.

#### 7.2.2.4 udGetDnsParams

##### Prototype

```
void udGetDnsParams (
    NQ_WCHAR *domain
    NQ_WCHAR *server
);
```

##### Description

Read DNS parameters from project-specific resource

##### Parameters

domain	OUT	Pointer to the buffer for DNS domain name. This buffer's length is 64 NQ_WCHAR characters and this should include terminating zero. DNS domain is a fully-qualified domain name.
server	OUT	Pointer to the buffer for a list of DNS IP addresses separated by a semicolon symbol. This buffer's length is defined as UD_DNS_SERVERSTRINGSIZE (see 5.1.3) which is calculated in compile time. The number of DNS servers in the list should not exceed UD_NQ_MAXDNSSERVERS (see 7.3.2.2). DNS servers exceeding this number will be ignored.

##### Return Value

None

##### Notes



### 7.2.2.5 udGetWins

**Prototype**

```
NQ_IPADDRESS4 udGetWins(  
    void  
);
```

**Description**

Read WINS address

**Parameters**

None

**Return Value**

WINS IP address in Network Byte Order (NBO) or zero if no WINS is specified.

**Notes**

When WINS address is supplied by the UD level, NQ NetBIOS treats the target node as H (Hybrid). When there is no WINS, the node is treated as a B (Broadcast) node.

### 7.2.2.6 udGetDomain

#### Prototype

```
void udGetDomain(  
    NQ_WCHAR *buffer,  
    NQ_BOOL *isWorkgroup  
);
```

#### Description

Read domain name for the target host

#### Parameters

buffer	OUT
	Pointer to buffer for domain name. This buffer is 17 bytes long including terminating zero.
isWorkgroup	OUT
	Pointer to flag indicating if the returned name is workgroup.

#### Return Value

None

#### Notes

NQ calls this function once during the start up.

### 7.2.2.7 udGetTransportPriority

**Prototype**

```
NQ_INT udGetTransportPriority(  
    NQ_UINT transport  
);
```

**Description**

Query project level for transport priority

**Parameters**

transport                      IN

Transport ID:

1 – NetBIOS

2 – IPv4

3 – IPv6

**Return Value**

Priority value. A transport with higher value of priority will be used first. A number, returned by a call to this function has only relative meaning. Priority value 0 means that the corresponding transport is disabled.

**Notes**

This function determines the order in which NQ Client uses transports for connecting a remote server. NQ calls this function for each generated transport.

### 7.2.2.8 udGetServerComment

Prototype

```
void udGetServerComment (
    NQ_WCHAR *buffer
);
```

Description

Read server comment string from project-specific resource

Parameters

buffer	OUT
Pointer to buffer for server comment string. This buffer is 100 bytes long including terminating zero.	

Return Value

None

Notes

NQ calls this function only once.

### 7.2.2.9 udGetDriverName

Prototype

```
void udGetDriverName(  
    NQ_CHAR *buffer  
);
```

Description

Read driver name for NQ Client from project-specific resource

Parameters

buffer	OUT
Pointer to buffer for driver name. This buffer is 100 bytes long including terminating zero.	

Return Value

None

Notes

**7.2.2.10 udGetTaskPriorities**

## Prototype

```
NQ_INT udGetTaskPriorities(  
    void  
);
```

## Description

Get priority for NQ tasks

## Parameters

None

## Return Value

Task priority as in integer value

## Notes

NQ uses this value to run its tasks

### 7.2.2.11 udGetCredentials

#### Prototype

```
NQ_BOOL udGetCredentials(
    const void *uri,
    NQ_WCHAR *username,
    NQ_WCHAR *password,
    NQ_WCHAR *domain
);
```

#### Description

Get user credentials for specific network resource (URI)

#### Parameters

uri	IN	
		Path to a remote share
username	OUT	
		Pointer to buffer for user (account) name. This buffer 100 bytes long including terminating zero.
password	OUT	
		Pointer to buffer for password. This buffer size is defined in UD_CC_MAXPWDLEN (see 5.1.3) and includes a terminating zero.
domain	OUT	
		Pointer to buffer for domain. This buffer is 17 bytes long including terminating zero.

#### Return Value

TRUE on success, FALSE on failure

#### Notes

NQ calls this function when it is about to connect to a remote share. On call, NQ provides share path. UD response with user name, password and domain NQ should use when connecting to this share.

### 7.2.2.12 **udGetNextShare**

#### Prototype

```
NQ_BOOL udGetNextShare (
    NQ_WCHAR *shareName,
    NQ_WCHAR *sharePath,
    NQ_BOOL* printQueue,
    NQ_WCHAR *shareDescription
);
```

#### Description

Enumerate list of shares

#### Parameters

shareName	OUT
	Pointer to buffer for share name. This buffer's length is UD_FS_MAXSHARELEN (see 7.3.1) including terminating zero.
sharePath	OUT
	Pointer to buffer for local path. This buffer's length is UD_FS_MAXPATHLEN (see 7.3.1) including terminating zero.
printQueue	OUT
	Specifies whether the share is a printer share
shareDescription	OUT
	Pointer to buffer for share description. This buffer's length is UD_FS_MAXDESCRIPTIONLEN (see 7.3.1) including terminating zero.

#### Return Value

TRUE when another share was responded, FALSE when no more shares are available

#### Notes

NQ repeatedly calls this function on Server start until it returns FALSE. Each call defines one share.

A share with terminating '\$' sign is treated as hidden. This share will not be reported to the client. Also a client will not be able to implicitly connect to it. It is recommended to always define at least one hidden share (e.g., - C\$), since this share will enable the functionality described in section 3.2.2 in [1] . Multiple hidden shares can be optionally defined.



### 7.2.2.13 udGetNextShareEx

#### Prototype

```
NQ_BOOL udGetNextShareEx(
    NQ_WCHAR* name,
    NQ_WCHAR* map,
    NQ_BOOL* isPrinter,
    NQ_WCHAR* description,
    NQ_BOOL* isEncrypted
);
```

#### Description

Enumerate list of shares (extended)

#### Parameters

name	OUT	Pointer to buffer for share name. This buffer's length is UD_FS_MAXSHARELEN (see 7.3.1) including terminating zero.
map	OUT	Pointer to buffer for the map path. This buffer's length is UD_FS_MAXPATHLEN (see 7.3.1) including terminating zero.
isPrinter	OUT	Pointer to variable getting FALSE for file system share and TRUE for a print queue.
description	OUT	Pointer to buffer for share description. This buffer's length is UD_FS_MAXDESCRIPTIONLEN (see 7.3.1) including terminating zero.
isEncrypted	OUT	pointer to variable getting FALSE for regular share and TRUE for encrypted share (SMB3).

#### Return Value

TRUE when another share was responded, FALSE when no more shares are available

#### Notes

### 7.2.2.14 udSaveShareInformation

#### Prototype

```
NQ_BOOL
udSaveShareInformation(
    const NQ_WCHAR* name,
    const NQ_WCHAR* newName,
    const NQ_WCHAR* newMap,
    const NQ_WCHAR* newDescription
);
```

#### Description

Add/Modify a share in the persistent store

#### Parameters

name	IN	Share name pointer. If this value is NULL – a new share should be created. If this value points to a name – a share should be modified.
newName	IN	Pointer to a new name for this share.
newMap	IN	Pointer to a new path for this share
newDescription	IN	Pointer to a new share description

#### Return Value

TRUE to approve creation/modification of the share, FALSE to cancel share creation/modification

#### Notes

The implementation of this function should return TRUE in the following cases:

- New share was persistently stored
- Existing share was persistently modified
- New share was not persistently stored but its creation was approved anyway. Such share will be exposed until server shutdown.
- An existing share was not persistently modified but this modification was approved anyway. This share will expose modified information until server shutdown.

This routine should return FALSE when a new share should not be created or an existing share should not be modified.

### 7.2.2.15 udRemoveShare

## Prototype

```
NQ_BOOL
udRemoveShare(
    const NQ_WCHAR* name
);
```

### Description

## Remove share from the persistent store

## Parameters

```
name                                IN
                                   Share name pointer.
```

### Return Value

TRUE when removal operation was approved, FALSE otherwise

## Notes

When this function returns TRUE it does not necessary mean that the share was removed from the persistent store.

**7.2.2.16 udGetFileSystemName**

## Prototype

```
void udGetFileSystemName(  
    const NQ_WCHAR *shareName,  
    const NQ_WCHAR *sharePath,  
    NQ_WCHAR *fileSystemName  
);
```

## Description

Enumerate list of shares

## Parameters

shareName	IN
Pointer to the share name.	
sharePath	IN
Pointer to the share path	
fileSystemName	OUT
Pointer to buffer for file system name	

## Return Value

None

## Notes

This function uses either share name or share path to determine the type of the target filesystem. Then it copies the name of this file system into provided buffer. The name should not be longer than 100 characters.

**7.2.2.17 udGetNextMount**

## Prototype

```
NQ_BOOL udGetNextMount(  
    NQ_WCHAR *mountName,  
    NQ_WCHAR *mountPath  
);
```

## Description

Enumerate list of mounts

## Parameters

mountName	OUT
Pointer to buffer for mount name. This buffer's length is 64 bytes including terminating zero.	
mountPath	OUT
Pointer to buffer for local path. This buffer's length is 64 bytes including terminating zero.	

## Return Value

TRUE when another mount was responded, FALSE when no more mounts are available

## Notes

NQ repeatedly calls this function on Client start until it returns FALSE.

### 7.2.2.18 **udAllocateBuffer**

#### Prototype

```
NQ_BYTE *udAllocateBuffer(  
    NQ_INT index,  
    NQ_COUNT numBuffer,  
    NQ_UINT bufferSize  
);
```

#### Description

Allocate buffer in the user space

#### Parameters

index	IN
Buffer index zero based	
numBuffer	IN
Total number of buffers	
bufferSize	IN
Buffer size in bytes.	

#### Return Value

Pointer to the next buffer

#### Notes

NQ calls this function twice during NQ start. User may implement one of the following methods:

- return a pointer to statically allocated buffers (two buffers are required)
- get space from the dynamic memory (the heap)
- allocate buffer in special memory space (e.g., - non-cacheable memory)

**7.2.2.19 udReleaseBuffer**

## Prototype

```
void udReleaseBuffer(  
    NQ_INT index,  
    NQ_COUNT numBuffers,  
    NQ_BYTE *bufferAddress,  
    NQ_UINT bufferSize  
);
```

## Description

Release a buffer, allocated in the user space

## Parameters

index	IN
Buffer index zero based	
numBuffers	IN
Total number of buffers	
bufferAddress	IN
Pointer to the buffer to release	
bufferSize	IN
Buffer size in bytes.	

## Return Value

None

## Notes

- NQ calls this function twice during NQ shutdown. This call releases the same buffers that were allocated during *udAllocateBuffer()* call (see 7.2.2.18). The implementation of this method should reflect the implementation of *udAllocateBuffer()*.

**7.2.2.20 udGetPort**

## Prototype

```
NQ_PORT udGetPort(  
    NQ_PORT default  
);
```

## Description

Redefine port number

## Parameters

`default`                      IN  
Default port number

## Return Value

Port number to used instead of the default

## Notes

NQ uses the following ports:

- Port 137 - NetBIOS Naming Service listener
- Port 138 - NetBIOS Datagram Service listener
- Port 139 – NetBIOS Session Service listener
- Internal ports defined in 7.3.1.14, 7.3.1.15 and 7.3.1.16

For coexistence with other network components NQ allows to redefine any of those ports by means of this function. NQ calls it each time a port is about to use, providing it with a default value. NQ uses the return value as an actual port number. The default implementation simply returns the *default* value thus no redefinition is performed. For redefining a port, modify this function so that it will return an alternate port number for the default port of the interest. For other ports that should not be alternated this function should continue to return its only argument.



### 7.2.2.21 udGetComputerId

## Prototype

```
VOID udGetComputerId(
    NQ_BYTE* buffer
);
```

### Description

## Get a unique computer ID

## Parameters

buffer	OUT
12 – byte buffer for a computer ID	

### Return Value

None

## Notes

The returned 12-byte value should be:

- "statistically unique" for the given machine
- persistently the same for each call

Recommended methods are:

- MAC address of the default adapter
- product serial number when available

**7.2.2.22 udGetMessageSigningPolicy**

## Prototype

```
NQ_BOOL udGetMessageSigningPolicy(  
    NQ_INT* policy  
);
```

## Description

Get server message signing policy

## Parameters

policy	OUT
--------	-----

Pointer to parameter, values are:  
0 - disabled  
1 - enabled  
2 - required

## Return Value

TRUE on success, FALSE on failure

## Notes

### 7.2.2.23 **udGetGlobalEncryption**

#### Prototype

```
NQ_BOOL udGetGlobalEncryption(  
    void  
);
```

#### Description

Get server global encryption

#### Parameters

None

#### Return Value

TRUE - global encryption  
FALSE - no global encryption

#### Notes

This setting is controlled by UD only, so when parameter not found in udconfig.c it means FALSE.

**7.2.2.24 udGetInternalCapture**

## Prototype

```
NQ_BOOL udGetInternalCapture(  
    void  
);
```

## Description

Whether to use internal capture

## Parameters

None

## Return Value

TRUE - use internal capture  
FALSE - do not use internal capture

## Notes

Returns FALSE only when parameter was present and FALSE in udconfig.c file, parameter not present in udconfig.c file is considered as TRUE, meaning default. NQ core setting is not affected.

**7.2.2.25 udGetInternalTrace**

## Prototype

```
NQ_BOOL udGetInternalTrace(  
    void  
);
```

## Description

Whether to use internal trace

## Parameters

None

## Return Value

TRUE - use internal trace  
FALSE - do not use internal trace

## Notes

Returns FALSE only when parameter was present and FALSE in udconfig.c file, parameter not present in udconfig.c file is considered as TRUE, meaning default. NQ core setting is not affected.

**7.2.2.26 udGetHostName**

## Prototype

```
void udGetHostName(  
    NQ_CHAR* buffer,  
    NQ_UINT length  
);
```

## Description

Get host name

## Parameters

buffer	OUT
Pointer to buffer for host name.	
length	IN
Length of the buffer.	

## Return Value

None

## Notes

### 7.2.2.27 udSetCredentials

#### Prototype

```
Void udSetCredentials(  
    const char* user,  
    const char* pwd,  
    const char* domain  
);
```

#### Description

Set new client credentials

#### Parameters

user	IN
User name for client login.	
pwd	IN
Password for client login.	
domain	IN
Domain name.	

#### Return Value

None

#### Notes

In order to use this function properly, the format should be as follows:  
udSetCredentials("username", "password", "domain").  
It is possible to omit the fields, then, the default hardcoded fields will be used.

**7.2.2.28**



### 7.2.2.29 udGetCaptureFileBaseFolder

#### Prototype

```
void udGetCaptureFileBaseFolder(
    NQ_CHAR * buffer,
    NQ_UINT  size
);
```

#### Description

Get base folder path of internal capture file

#### Parameters

buffer	OUT
Buffer to assign the path to.	
size	IN
Maximun characters to assign into buffer.	

#### Return Value

None

#### Notes

None

**7.2.2.30      udGetLogFileBaseFolder**

## Prototype

```
void udGetLogFileBaseFolder(  
    NQ_CHAR * buffer,  
    NQ_UINT  size  
);
```

## Description

Get base folder path of log file

## Parameters

buffer	OUT
Buffer to assign the path to.	
size	IN
Maximun characters to assign into buffer.	

## Return Value

None

## Notes

None

### 7.2.3 Domain Membership

Functions in this group are used to provide project-dependent support for Domain Membership mechanism. They are only required when UD\_CC\_INCLUDEDOMAINMEMBERSHIP (see 7.3.1.39).

### 7.2.3.1 udGetComputerSecretByDomain

#### Prototype

```
NQ_BOOL udGetComputerSecretByDomain (
    NQ_BYTE *secret,
    const NQ_WCHAR *domainDNS,
    NQ_WCHAR *domainNB
);
```

#### Description

Read security descriptor

#### Parameters

<code>secret</code>	OUT
Buffer for a secret pointer. If a value inside that buffer is non\-\NULL, this function should fill this buffer with a pointer to computer secret.	
<code>domainDNS</code>	IN
Domain name DNS	
<code>domainNB</code>	OUT
Domain name NetBIOS. If it is NULL, the function checks for computer secret which refers only to domainDNS	

#### Return Value

if the computer secret is available and (if the secret parameter was not NULL) was copied into the buffer and FALSE when the computer secret is not available for one of the following reasons:

- \* NQ did not join the domain yet.
- \* The computer secret was not properly saved.
- \* The computer secret is not accessible.

#### Notes

NQ assumes that computer secret is persistent and was created in a call to udSetComputerSecretByDomain (see 7.2.3.2).

### 7.2.3.2 udSetComputerSecretByDomain

#### Prototype

```
void udSetComputerSecretByDomain (  
    const NQ_BYTE *secret,  
    const NQ_WCHAR *domainDNS,  
    const NQ_WCHAR *domainNB  
);
```

#### Description

Read security descriptor

#### Parameters

secret	IN
Pointer to computer secret.	
domainDNS	IN
Domain name DNS	
domainNB	IN
Domain name NetBIOS. If it is NULL, the function checks for computer secret which refers only to domainDNS	

#### Return Value

None.

#### Notes

NQ assumes that computer secret is persistent and after this call it will be available through a call to udGetComputerSecretByDomain (see 7.2.3.1).

#### 7.2.4 Event Log

On designated events NQ core software calls so-called user-defined event log function – *udEventLog()*. NQ software provides a placeholder implementation for this function while it is user's responsibility to implement actual event log in place of this function.

Function *udEventLog()* is called in the current task context. For instance, CIFS/SMB server task (SMB) calls this function in its own context. Synchronization of event logs for multiple tasks is out of the NQ scope.

NQ core calls *udEventLog()* with the only information that is not available outside NQ software. Time-stamping, for instance, is out of the NQ event logging scope.

The following event logging functionality is out of NQ scope and, whether required, should be provided by user application software:

- Storing event log in a file or any persistent destination
- Temporary enabling/disabling even log
- Time-stamping
- Human-friendly representation of event log
- Event filters

An application, build on top of NQ software can be compiled with or without event logging capability. This option is controlled by *UD\_NQ\_INCLUDEEVENTLOG* parameter found in the *udparams.h* file (see 7.3.1.23). When such application is compiled without event logging, NQ core software does not call *udEventLog()*.

All string values, delegated by NQ core to the event logging level are in *NQ\_WCHAR* character encoding. Actual type of *NQ\_WCHAR* depends on *UD\_CM\_UNICODEAPPLICATION* (see 7.3.1.6).

### 7.2.4.1 udEventLog

#### Prototype

```
void udEventLog(
    NQ_UINT module,
    NQ_UINT class,
    NQ_UINT type,
    const NQ_WCHAR *username,
    NQ_IPADDRESS *ip,
    NQ_UINT32 status,
    const NQ_BYTE *parameters
);
```

#### Description

This function is called by NQ to log an event.

#### Parameters

module	IN	NQ module that originated this event (see 5.1.2) as: UD_LOG_MODULE_SERVER or UD_LOG_MODULE_CLIENT
class	IN	event class (see 5.1.2) as: UD_LOG_CLASS_GEN, UD_LOG_CLASS_FILE or UD_LOG_CLASS_SHARE
type	IN	event type (see 5.1.2)
userName	IN	Pointer to the user name string. For UD_LOG_CLASS_GEN class events this value is NULL
ip	IN	Pointer to the IP address on the second side of the connection. For UD_LOG_CLASS_GEN class events this value is NULL
status	IN	This value is zero if the operation has succeeded. It contains error code if the operation has failed. For an NQ Server event this code is the same that will be transmitted to the client and it has so-called DOS format. For an NQ client event this value is the same that will be installed as system error (e.g., - <i>errno</i> ) on return from the operation.
parameters	IN	Pointer to a structure that is filled with event data. Actual structure depends on event type. See 5.2 for available structures. For those events that do not have parameters

(e.g., - events of class UD\_LOG\_CLASS\_GEN) this value may be NULL.

#### Return Value

None

#### Notes

Function *udEventLog()* is called in the current task context. For instance, CIFS/SMB server task (SMB) calls this function in its own context. Synchronization of event logs for multiple tasks is out of the NQ scope.

The following event logging functionality is out of NQ scope and, whether required, should be provided by user application software:

- Storing event log in a file or any persistent destination
- Temporary enabling/disabling even log
- Time-stamping
- Human-friendly representation of event log
- Event filters

An application, build on top of NQ software can be compiled with or without event logging capability. This option is controlled by *UD\_NQ\_INCLUDEEVENTLOG* parameter found in the *udparams.h* file. When such application is compiled without event logging, NQ core software does not call *udEventLog()*.

All string values, delegated by NQ core to the event logging level are in *NQ\_WCHAR* character encoding. Actual type depends on the *UD\_CM\_UNICODEAPPLICATION* parameter (see [1] and 7.3.1.6).



### 7.2.5 Error Code Conversions

Functions in this group perform conversions between different sets of error codes used in NQ:

1. System error code. This is how the operating system reports of the last error. The POSIX analog is *errno*.
2. NQ internal codes are used by NQ Client to distinguish between different error conditions.

### 7.2.5.1 udGetSmbError

## Prototype

```

NQ_UINT32 udGetSmbError(
    NQ_UINT32 sysErr
);

```

### Description

## Convert last system error to SMB error code

## Parameters

sysErr	IN
Last system error	

## Return Value

SMB status code in DOS format (see Notes) or zero to use project-independent conversion

## Notes

The return value is a 32-bit integer values composed by the following formula:

$$\langle \text{return value} \rangle = 0x10000 * \langle \text{code} \rangle + \langle \text{class} \rangle$$

Where:

*<class>* - error class

<code> - error code

For error classes and error codes in DOS format see [3].

This call is intended to withdraw the last system error and to convert it into the most appropriate SMB error. Function *syGetLastSmbError* (see 6.4.1.11.2) may first call this function to perform an optional project-dependent conversion. If this function returns zero, it is *syGetLastSmbError* that performs code conversion.

### 7.2.5.2 udNqToSystemError

**Prototype**

```
NQ_INT udNqToSystemError(  
    NQ_UINT32 nqErr  
);
```

**Description**

Convert NQ error to last system error

**Parameters**

nqErr	IN
Last NQ error	

**Return Value**

System error code or zero if project-independent code conversion should be used

**Notes**

Implementation should convert NQ error into system error code (for instance – POSIX error codes). This function is complimentary and may be used for application convenience. It is not used by NQ core software. User may place here a proprietary conversion table. .

### 7.2.6 Code Page Conversion

NQ software is capable of converting Unicode to ANSI and back with a one set of national characters. This set is called a *codepage*. Since NQ software supports one codepage in a time, it is application's responsibility to define the codepage.

The shipped version of these functions supports an "empty" codepage for US English.

### 7.2.6.1 udGetCodePage

Prototype

```
NQ_INT udGetCodePage(  
    void  
);
```

Description

Get the current codepage number.

Parameters

None.

Return Value

Codepage number according to Microsoft OEM code pages supported by Windows.

Notes

See <http://www.microsoft.com/globaldev/reference/oem.msp>

### 7.2.6.2 udSetCodePage

Prototype

```
void udSetCodePage(  
    NQ_INT codePage  
);
```

Description

Set the current code page number. NQ converts Unicode to ANSI and vice versa according to the current code page. This function gives the ability to change the current code page in run time, but usually it is called only in initialization.

Parameters

Code page number.

Return Value

None.

Notes

See <http://www.microsoft.com/globaldev/reference/oem.msp>

### 7.2.7 Activate or deactivate SMB 1 dialect

SMB 1 dialect can be disabled/enabled by comment/uncomment the macro “UD\_NQ\_INCLUDESMB1” (exclude SMB 1 client files from build) or by “SUPPORTSMB1” parameter in client configuration file. SMB1 can also be disabled/enabled in run time using Clinet API functions ccCifsGetSMB1Support() and ccCifsSetSMB1Support().

### 7.2.7.1 udDefGetClientSMB1Support

#### Prototype

```
NQ_Bool udDefGetClientSMB1Support(  
    void  
);
```

#### Description

This function is called in client startup to initialize SMB 1 support for client according to “SUPPORTSMB1” parameter from client configuration file. If this parameter does not present in client configuration file it is considered as TRUE.

#### Parameters

None

#### Return Value

TRUE or FALSE.

#### Notes

None

### **7.3 *Compile-time Parameters***

UD module defines compile-time parameters for the core NQ code. This option is available with source code license only. UD compile-time parameters reside in file UDPARAMS.H.

#### **7.3.1 Functional Parameters**

These parameters control NQ behavior.



### 7.3.1.1 UD\_NQ\_INCLUDETRACE

Prototype

UD\_NQ\_INCLUDETRACE

Description

Enable/disable trace log.

Notes

With this parameter defined NQ generate trace logs as described in 6.4.4. When this parameter is commented, trace logs are not generated.

.

### 7.3.1.2 UD\_NQ\_USETRANSPORTNETBIOS

Prototype

UD\_NQ\_USETRANSPORTNETBIOS

Description

Generate NQ with NetBIOS support

Notes

With this parameter defined NQ Client attempts to connect remote server using NetBIOS services (NBNS and NBSS). NQ Core determines the priority of this transport by calling *upGetTransportPriority()* (see 7.2.2.7).

### 7.3.1.3 UD\_NQ\_USETRANSPORTIPV4

Prototype

UD\_NQ\_USETRANSPORTIPV4

Description

Generate NQ with IPv4 support

Notes

With this parameter defined NQ Client resolves remote server name over DNS and attempts to connect it on port 445. NQ Core determines the priority of this transport by calling *upGetTransportPriority()* (see 7.2.2.7). This transport is also known as “naked CIFS”.

#### 7.3.1.4 UD\_NQ\_USETRANSPORTIPV6

Prototype

UD\_NQ\_USETRANSPORTIPV6

Description

Generate NQ with IPv6 support

Notes

With this parameter defined NQ Client resolves remote server name over DNS and attempts to connect it on port 445 using IPv6 address scheme. NQ Core determines the priority of this transport by calling *upGetTransportPriority()* (see 7.2.2.7).

### 7.3.1.5 UD\_NQ\_INCLUDESMBCAPTURE

Prototype

UD\_NQ\_INCLUDESMBCAPTURE

Description

Generates internal network capture file (.pcap)

Notes

With this parameter defined NQ will Create a .pcap file which will contain the received and sent SMB/SMB2/SMB3 messages. In the case that SMB3 encrypted messages are received/sent a decrypted message will be written in the .pcap file.

### 7.3.1.6 UD\_CM\_UNICODEAPPLICATION

Prototype

UD\_CM\_UNICODEAPPLICATION

Description

Whether the default API is Unicode

Notes

This parameter has two meaning:

NQ exposes three application interfaces: 1) ANSI only, 2) Unicode only and 3) default. While first two interfaces are explicit, the default one may be implicitly ANSI or implicitly Unicode. This parameter, when defined, makes the default interface Unicode. When it is commented, the default interface is ANSI.

This parameter affects also the definition of NQ\_WCHAR (see [1] and [2] ). NQ\_WCHAR is a data type for storing names inside NQ. This includes, first of all, filenames. When this parameter is defined, all names are stored internally in Unicode. Otherwise, they are stored in ANSI.

### 7.3.1.7 UD\_CM\_DONOTREGISTERHOSTNAME

Prototype

UD\_CM\_DONOTREGISTERHOSTNAME

Description

Whether to skip host name registration (deprecated)

Notes

Starting from NQ 7.00 this parameter has no effect. Parameters 7.3.1.8 and 7.3.1.9 are used instead.

### 7.3.1.8 UD\_CM\_DONOTREGISTERHOSTNAMEDNS

Prototype

UD\_CM\_DONOTREGISTERHOSTNAMEDNS

Description

Whether to skip host name registration on DNS

Notes

On NQ startup, it registers its host name over both NetBIOS and DNS.

This behavior can be omitted if a vendor, using NQ decides on a policy which prevents his device from host name registration. When the described parameter is defined, NQ skips name registration on DNS. This parameter does not affect name registration over NetBIOS.

This parameter is intended for future use and it should not be defined in NQ 7.00



### 7.3.1.9 UD\_CM\_DONOTREGISTERHOSTNAMENETBIOS

Prototype

UD\_CM\_DONOTREGISTERHOSTNAMENETBIOS

Description

Whether to skip host name registration over NETBIOS

Notes

On NQ startup, it registers its host name over both NetBIOS and DNS.

This behavior can be omitted if a vendor, using NQ decides on a policy which prevents his device from host name registration. When the described parameter is defined, NQ skips name registration over NETBIOS. This parameter does not affect name registration on DNS.

This parameter is intended for future use and it should not be defined in NQ 7.00

### 7.3.1.10 UD\_NQ\_INCLUDECODEPAGE

#### Prototype

UD\_NQ\_INCLUDECODEPAGE

#### Description

Whether NQ supports code page conversion

#### Notes

When this parameter is not defined NQ supports only English Unicode to ANSI and ANSI to Unicode conversion. NQ converts Unicode to ANSI by taking the first byte of the UTF-16LE code and masking it by 0x7F. It converts from ANSI to Unicode by placing one-byte ANSI code into the first byte of UFT-16LE code and zeroing the second byte.

When this parameter is defined NQ expects also that:

- Function *udGetCodePage()* (see 7.2.6.1) returns the current code page number as <N>, e.g. 932 for Shift-JIS Japanese encoding.
- Function *udSetCodePage()* (see 7.2.6.2) sets the current code page number.
- Parameter UD\_NQ\_CODEPAGE<N> is also defined in file *udparams.h*, e.g. UD\_NQ\_CODEPAGE932. When code page support (UD\_NQ\_INCLUDECODEPAGE) is defined then UD\_NQ\_CODEPAGE437 must also be defined.
- NQ core supports codepage <N>. Currently NQ supports the following code pages:
  - 437 – US
  - 850 – Multilingual Latin I
  - 852 – Latin II
  - 858 – Multilingual Latin I + Euro
  - 862 – Hebrew
  - 932 – Japanese Shift-JIS
  - 936 – Simplified Chinese GBK
  - 949 – Korean
  - 950 – Traditional Chinese Big5
  - 8 – UTF-8

**7.3.1.11 UD\_NS\_SCOPEIDLEN**

## Prototype

`UD_NS_SCOPEIDLEN`

## Description

Maximum length of scope ID string including the terminating zero

## Notes

NQ uses only one buffer for scope ID

**7.3.1.12    UD\_NS\_MAXADAPTERS**

## Prototype

`UD_NS_MAXADAPTERS`

## Description

Maximum number of network adapters (host IP addresses) that NQ supports

## Notes

NQ requires approximately 80 bytes of RAM per adapter. This parameter retains its name for compatibility with earlier versions. However, starting from version 7.00 this parameter designates a maximum number of host IP addresses rather than a number of adapters. For instance, when host has two adapters with two IPv4 addresses and one IPv6 address for each of them, this value should be six at least.

**7.3.1.13 UD\_NS\_NUMSOCKETS**

## Prototype

`UD_NS_NUMSOCKETS`

## Description

Maximum number of sockets of all types

## Notes

This number includes server socket, accepted sockets and connected client sockets and it does not include internal sockets (see 7.3.2.1 and 7.3.2.1).

NQ requires approximately 80 bytes per socket.

This number may be calculated as: <number of concurrent server sessions> + <number of hosts NQ Client is connected to> + 1.

This number should not exceed the number of underlying sockets.

The default value is 35

**7.3.1.14    UD\_BR\_INTERNALIPCPORT**

## Prototype

`UD_BR_INTERNALIPCPORT`

## Description

Number of port for internal communications with Browser Daemon

## Notes

NQ NetBIOS uses internal ports for communications with other NQ modules. The value of this parameter should not contradict with other ports in use on the target. Usually, the default value is sufficient. To change this port number in run-time use function 7.2.2.20.

**7.3.1.15      UD\_NS\_INTERNALNSPORT**

## Prototype

`UD_NS_INTERNALNSPORT`

## Description

Number of port for internal communications with NetBIOS Naming Service

## Notes

NQ NetBIOS uses internal ports for communications with other NQ modules. The value of this parameter should not contradict with other ports in use on the target. Usually, the default value is sufficient. To change this port number in run-time use function 7.2.2.20.

**7.3.1.16      UD\_NS\_INTERNALDSPORT**

## Prototype

`UD_NS_INTERNALDSPORT`

## Description

Number of port for internal communications with NetBIOS Datagram Service

## Notes

NQ NetBIOS uses internal ports for communications with other NQ modules. The value of this parameter should not contradict with other ports in use on the target. Usually, the default value is sufficient. To change this port number in run-time use function 7.2.2.20.



**7.3.1.17 UD\_NB\_BINDBROADCASTS**

## Prototype

`UD_NB_BINDBROADCASTS`

## Description

Whether to use separate sockets for broadcasts

## Notes

In a standard implementation of BSD sockets a bound socket gets both unicasts and broadcasts. In this case binding socket to a broadcast address is not necessary and even is sometimes an illegal operation.

However, in some implementations a bound socket does not get broadcasts. There we need two separate sockets - one for broadcasts and another one for unicasts.

Define the following parameter if two separate sockets are required.

**7.3.1.18      UD\_NQ\_AVOIDDCRESOLUTIONNETBIOS**

## Prototype

`UD_NQ_AVOIDDCRESOLUTIONNETBIOS`

## Description

Whether to use NetBIOS name resolution for resolving DC

## Notes

When NetBIOS is defined, NQ may use it for resolving domain controllers (DC). With a B-node this resolution may slow down the entire DC resolution because of a broadcast response timeout. It becomes time-costly when DFS is enabled. This parameter, when defined, forces NQ to skip DC resolution over NetBIOS even when NetBIOS is available.

### **7.3.1.19      UD\_NB\_RETARGETSESSIONS**

Prototype

UD\_NB\_RETARGETSESSIONS

Description

Whether NQ NetBIOS will retarget session requests

Notes

NetBIOS may support more than one server application on the target. For this reason, Session Request message should be accepted by the NetBIOS Daemon and retargeted to the listening port of a respective server. This requires from connecting CIFS/SMB clients to understand Session Retarget packages.

If this parameter is defined NQ will use SESSION RETARGET.

When this parameter is defined it requires UD\_ND\_INCLUDENBDAEMON (7.3.1.21) to be also defined. If this condition is not satisfied there will be an error during preprocessing.

**7.3.1.20 UD\_NB\_CHECKCALLEDNAME**

## Prototype

`UD_NB_CHECKCALLEDNAME`

## Description

Whether NQ NetBIOS will check the called name in SESSION REQUEST

## Notes

Usually NetBIOS checks that a SESSION REQUEST message was send to a known name (the name of the host computer or SMB alias - \*SMBSEV or IP address).

In some projects CIFS/SMB should answer to multiple names, not necessarily the same as the host name. In this case it does not check the called name in a SESSION REQUEST packet.

To exclude name check the definition of UD\_NB\_CHECKCALLEDNAME should be commented.

This parameter may be omitted when external Name Service is used (see 7.3.1.22).

In order to disable IP address check, check UD\_NB\_CHECKCALLEDNAME\_NOIP 7.3.1.54)

**7.3.1.21 UD\_ND\_INCLUDENBDAEMON**

## Prototype

`UD_ND_INCLUDENBDAEMON`

## Description

Whether to generate NQ NetBIOS

## Notes

Some projects utilize external (non NQ) NetBIOS implementation. In this case NQ should be compiled without NetBIOS. When this definition is omitted (commented) the following definitions should be omitted as well:

`UD_NB_INCLUDENAMESERVICE (7.3.1.22),``UD_NB_RETARGETSESSIONS (7.3.1.17)`

If this condition is not satisfied there will be an error during preprocessing.

**7.3.1.22    UD\_NB\_INCLUDENAMESERVICE**

## Prototype

`UD_NB_INCLUDENAMESERVICE`

## Description

Whether NQ NetBIOS includes NetBIOS Name Service

## Notes

Some projects may contain external (non NQ) NetBIOS Naming Service - NBNS. In this case NQ should be compiled without NBNS. When this is required the definition of `UD_NB_INCLUDENAMESERVICE` should be commented. When this parameter is defined it requires `UD_ND_INCLUDENBDAEMON` (7.3.1.21) to be also defined. If this condition is not satisfied there will be an error during preprocessing.

### **7.3.1.23    UD\_NB\_INCLUDELLMNRRESPONDER**

Prototype

UD\_NB\_INCLUDELLMNRRESPONDER

Description

Whether NQ NetBIOS Daemon includes IPv4 LLMNR responder

Notes

Some platforms may contain external (non NQ) LLMNR responder. In this case it requires either to disable the external responder or to compile NQ NetBIOS Daemon without this parameter. If there is external LLMNR responder and NQ LLMNR responder is enabled too, NQ NetBIOS Daemon will not able to start. When this parameter is defined it requires UD\_ND\_INCLUDENBDAEMON (7.3.1.21) and UD\_NQ\_USETRANSPORTIPV4 (7.3.1.3) to be also defined. If this condition is not satisfied there will be an error during preprocessing.

**7.3.1.24 UD\_NQ\_INCLUDEEVENTLOG**

## Prototype

`UD_NQ_INCLUDEEVENTLOG`

## Description

Whether NQ calls event log function

## Notes

When this parameter is defined NQ calls *udEventLog* on designated events. See 7.2.4.1 for more details.



**7.3.1.25 UD\_FS\_MAXSHARELEN**

Prototype

UD\_FS\_MAXSHARELEN

Description

Buffer lengths for share name

Notes

This value applies to validate share name.

**7.3.1.26 UD\_FS\_MAXPATHLEN**

Prototype

UD\_FS\_MAXPATHLEN

Description

Buffer lengths for a name of share path

Notes

None

**7.3.1.27 UD\_FS\_MAXDESCRIPTIONLEN**

Prototype

UD\_FS\_MAXDESCRIPTIONLEN

Description

Buffer lengths for share description

Notes

This value applies to share table in NQ Server

**7.3.1.28      UD\_NQ\_INCLUDESMB1**

## Prototype

```
UD_NQ_INCLUDESMB1
```

## Description

Whether to generate SMB (SMB1) support.

## Notes

When this parameter is defined, NQ supports SMB (SMB1) protocol unless it is disabled by other mechanism (the API function `ccCifsSetSMB1Support()` or the parameter “SUPPORTSMB1” in client configuration file). When this parameter is commented, NQ supports only SMB2, SMB3 and SMB3.1.1.  
Compiling NQ without SMB1 grants less footprint.  
This parameter affects both NQ Client and NQ Server.

**7.3.1.29 UD\_NQ\_INCLUDESMB2**

## Prototype

```
UD_NQ_INCLUDESMB2
```

## Description

Whether to generate SMB2 support.

## Notes

When this parameter is defined, NQ supports both SMB and SMB2 protocols. When this parameter is commented, NQ supports SMB only.

Compiling NQ without SMB2 grants less footprint.

This parameter effects on NQ Client.

**7.3.1.30      UD\_NQ\_INCLUDESMB3**

## Prototype

```
UD_NQ_INCLUDESMB3
```

## Description

Whether to generate SMB3 support.

## Notes

When this parameter is defined, NQ supports both SMB and SMB2 and SMB3 protocols. When this parameter is commented, NQ supports only SMB and SMB2.  
Compiling NQ without SMB3 grants less footprint.  
This parameter effects on NQ Client.

**7.3.1.31      UD\_NQ\_INCLUDESMB311**

## Prototype

```
UD_NQ_INCLUDESMB311
```

## Description

Whether to generate SMB3.1.1 support.

## Notes

When this parameter is defined, NQ supports both SMB, SMB2, SMB3 and SMB3.1.1 protocols. When this parameter is commented, NQ supports only SMB, SMB2 and SMB3.  
Compiling NQ without SMB3.1.1 grants less footprint.  
This parameter effects on NQ Client.

**7.3.1.32    UD\_NQ\_INCLUDECIFSCIENT**

## Prototype

`UD_NQ_INCLUDECIFSCIENT`

## Description

Whether to generate NQ client

## Notes

In order for NQ client to work properly it requires NetBIOS daemon presence in the system which means that the UD\_NB\_INCLUDENBDAEMON symbol (7.3.1.21) has to be defined.



**7.3.1.33      UD\_CC\_INCLUDEDFS**

## Prototype

`UD_CC_INCLUDEDFS`

## Description

Whether to perform DFS resolving

## Notes

When this definition is visible, NQ Client is compiled with DFS paths resolving. This, in turn, affects all API calls with file/directory/share name parameter. A developer, before including DFS support should consider the following:

- API calls involving file/directory/share names may work slowly then without DFS support. This additional delay is negligible on correct DFS links but it may be as much as 15 seconds on a broken link when the “failover” algorithm is applied.
- With DFS support NQ Client requires bigger stack. The amount of stack required differs depending on memory saving options (see 6.4.2.1.2) and internal character size (see 7.3.1.6). The additional stack requirements for a single iteration (see below) may be as much as 2KByte.
- DFS resolving algorithm is recursive. The number of iterations is currently limited to ten (while in real life it rarely exceeds three). An additional iteration applies the same stack requirements as above.

### 7.3.1.34 UD\_CC\_DFSDOMAINCACHESIZE

#### Prototype

```
UD_CC_DFSDOMAINCACHESIZE
```

#### Description

The dimension of DFS domain cache.

#### Notes

This definition has form of:

```
#define UD_CC_DFSDOMAINCACHESIZE <n>
```

When a domain is mentioned in a DFS path, NQ looks in the cache first, and only then it starts domain discovery process. NQ keeps track of the discovered domains for 24 hours. It can accommodate up to <n> entries. When a <n+1>s domain is discovered it replaces the latest entry in the cache.

This definition is optional. When not defined the domain cache is generated with five slots. This definition has effect only when DFS is included (see 7.3.1.33).

Each entry in this cache requires about 140 bytes of RAM.

See also ccDfsrefCleanup() in [2] .

**7.3.1.35 UD\_CC\_DFSSERVERCACHESIZE**

## Prototype

`UD_CC_DFSSERVERCACHESIZE`

## Description

The dimension of DFS server cache.

## Notes

This definition has form of:

```
#define UD_CC_DFSSERVERCACHESIZE <n>
```

When a server is mentioned in a DFS path, NQ looks in the cache first, and only then it starts server discovery process. NQ keeps track of the discovered servers for 24 hours. It can accommodate up to <n> entries. When a <n+1>s server is discovered it replaces the latest entry in the cache. The same cache is used for keeping Domain Controller hosts; therefore the recommended value of <n> should be greater than for domains (see 7.3.1.34).

This definition is optional. When not defined the server's cache is generated with ten slots. This definition has effect only when DFS is included (see 7.3.1.33).

Each entry in this cache requires about 140 bytes of RAM.

See also `ccDfsrefCleanup()` in [2] .

**7.3.1.36 UD\_CC\_DFSREFERRALCACHESIZE**

## Prototype

`UD_CC_DFSREFERRALCACHESIZE`

## Description

The dimension of DFS referral cache.

## Notes

This definition has form of:

```
#define UD_CC_DFSREFERRALCACHESIZE <n>
```

When resolving a path, NQ looks in the cache first, and only then it starts DFS referral discovery process. NQ keeps track of the discovered referrals for 24 hours. It can accommodate up to <n> entries. When a <n+1>s referral is about to be added NQ replaces the latest entry in the cache. The recommended value of <n> should be greater than for servers (see 7.3.1.35).

This definition is optional. When not defined the referral's cache is generated with 20 slots. This definition has effect only when DFS is included (see 7.3.1.33).

Each entry in this cache may require about 1060 bytes of RAM.

See also `ccDfsrefCleanup()` in [2] .

**7.3.1.37      UD\_CC\_INCLUDEEXTENDEDSECURITY**

## Prototype

`UD_CC_INCLUDEEXTENDEDSECURITY`

## Description

Whether NQ client supports extended security logons

## Notes

This parameter enables NQ Client to use extended security mechanism for user authentication.

Including Kerberos support (see 7.3.1.38) requires this parameter to be defined.

Although other authentication methods can work either with or without extended security it is strongly recommended to keep this parameter defined.

Un-defining this parameter is advisable to decrease the footprint, when NQ client is supposed to use LM and/or NTLM authentication methods only.

**7.3.1.38      UD\_CC\_INCLUDEEXTENDEDSECURITY\_KERBEROS**

## Prototype

`UD_CC_INCLUDEEXTENDEDSECURITY_KERBEROS`

## Description

Whether NQ client supports extended security logons using Kerberos 5 over GSSAPI over SASL

## Notes

The following parameter should be also defined:  
`UD_CC_INCLUDEEXTENDEDSECURITY` (see 7.3.1.37)

**7.3.1.39 UD\_CC\_INCLUDESECURITYDESCRIPTORS**

## Prototype

`UD_CC_INCLUDESECURITYDESCRIPTORS`

## Description

Whether NQ client supports file security descriptors

## Notes

This parameter allows NQ Client to set exclusive access rights for files and folders. See `ccSetExclusiveAccessToFile()` and `ccIsExclusiveAccessToFile()` in [2] .

**7.3.1.40      UD\_CC\_INCLUDEDOMAINMEMBERSHIP**

## Prototype

`UD_CC_INCLUDEDOMAINMEMBERSHIP`

## Description

Whether NQ client supports domain membership functionality

## Notes

This parameter enables the NQ Client functionally for joining, entering and leaving Microsoft Windows Domain. See `ccDomainJoin()`, `ccDomainLogon()` and `ccDomainLeave()` in [2] .



**7.3.1.41      UD\_CC\_INCLUDELDAP**

## Prototype

`UD_CC_INCLUDELDAP`

## Description

Whether NQ client supports Active Directory client functionality

## Notes

This parameter enables the NQ Client API for accessing Active Directory servers. Currently, NQ does not include LDAP client but rather provides an interface to external LDAP client implementation.

**7.3.1.42    UD\_NQ\_INCLUDEBROWSERDAEMON**

## Prototype

`UD_NQ_INCLUDEBROWSERDAEMON`

## Description

Whether to generate NQ network browser daemon

## Notes

This parameter is deprecated and is stated here for backwards compatibility only. Since NQ code does not use it anymore, this parameter may be removed. NQ does not use Browser Daemon anymore.

**7.3.1.43    UD\_CC\_CLIENTRETRYCOUNT**

## Prototype

`UD_CC_CLIENTRETRYCOUNT`

## Description

Parameter used to define the number of retries for the client

## Notes

When this parameter is defined NQ will use its value, and will retry any operation this number of times in case it fails (does not affect browsing)  
This definition is disabled by default.

**7.3.1.44    UD\_CC\_BROWSERRETRYCOUNT**

## Prototype

`UD_CC_BROWSERRETRYCOUNT`

## Description

Parameter used to define the number of retries for the client when browsing

## Notes

When this parameter is defined NQ will use its value, and will retry any browsing operation this number of times in case it fails. This definition is disabled by default.

**7.3.1.45 UD\_CM\_CAPTURE\_FILENAME**

## Prototype

`UD_CM_CAPTURE_FILENAME`

## Description

File name for capture output

## Notes

This parameter is only used when the  
UD\_NQ\_INCLUDESMBCAPTURE Macro is defined.

**7.3.1.46      UD\_CM\_SECURITYDESCRIPTORLENGTH**

## Prototype

`UD_CM_SECURITYDESCRIPTORLENGTH`

## Description

Parameter used to define the maximum length of a security descriptor in bytes

## Notes

UD\_NQ\_USETRANSPORTNETBIOS should be also defined.

**7.3.1.47      UD\_CM\_RESOLVERCACHENAMETTL**

## Prototype

`UD_CM_RESOLVERCACHENAMETTL`

## Description

Parameter used to define cached resolved host name TTL in seconds

## Notes

When this parameter is defined and equal to 0 then cache is disabled.

**7.3.1.48 UD\_CC\_INCLUDERPC**

Prototype

`UD_CC_INCLUDERPC`

Description

Compile flag for RPC support

Notes

None



**7.3.1.49      UD\_CC\_RPCDISABLECONNECTIONFALLBACKS**

## Prototype

`UD_CC_RPCDISABLECONNECTIONFALLBACKS`

## Description

When this flag is defined NQ will not attempt to connect remote RPC pipe using anonymous credentials or default credentials after it fails to connect using the provided credentials.

## Notes

Can be usefull to produce informative errors when attempt to connect RPC pipe was failed (using anonymous credentials might overwrite the last error with “access denied”).

**7.3.1.50 UD\_CC\_INCLUDEFSDRIVER**

Prototype

`UD_CC_INCLUDEFSDRIVER`

Description

Compile flag for FS driver support

Notes

None

**7.3.1.51 UD\_NQ\_CLOSESOCKETS**

## Prototype

`UD_NQ_CLOSESOCKETS`

## Description

Whether sockets are opened and closed per usage

## Notes

Various sockets, like name resolution sockets, can be either left open all the time or opened and closed per usage. When the following parameter is defined, sockets are opened and closed per usage. When software resides in untrusted network, it is advisable to enable the following parameter.

**7.3.1.52      UD\_CM\_INCLUDEWSDCLIENT**

## Prototype

`UD_NQ_INCLUDEWSDCLIENT`

## Description

Whether WS-Discovery client is used

## Notes

WS-Discovery protocol is used to obtain device information on local network, it discovers devices IPs (computers and printers). Used by NQ Client APIs to enumerate hosts on the network. Currently supported on Linux platforms.

**7.3.1.53 UD\_CM\_WSDCLIENTTIMEOUT**

Prototype

`UD_CM_WSDCLIENTTIMEOUT`

Description

Define WS-Discovery client timeout value

Notes

Time in milliseconds to wait for responses.

**7.3.1.54    UD\_NB\_CHECKCALLEDNAME\_NOIP**

Prototype

`UD_NB_CHECKCALLEDNAME_NOIP`

Description

Whether NQ NetBIOS will check the called IP in SESSION REQUEST

Notes

By default, when UD\_NB\_CHECKCALLEDNAME (7.3.1.20) is defined we check names, SMB alias and IP address when SESSION REQUEST arrives, For checking only Name and respond with a Negative response to IP address this define need to be enabled. To exclude IP check the definition of UD\_NB\_CHECKCALLEDNAME\_NOIP should be defined.

**7.3.1.55 UD\_CC\_INCLUDERPCOVERTCP**

## Prototype

`UD_CC_INCLUDERPCOVERTCP`

## Description

Compile flag for client's RPC over TCP support

## Notes

Currently supported Netlogon secure channel used for communication with Active Directory for pass through authentication of domain users. Default encryption used is AES (as most secure), less secure Strong Key and Des are not currently supported. When Active Directory doesn't support AES a fallback to pass through over SMB is implicitly done.

**7.3.1.56      UD\_CC\_NETLOGONENCRYPT**

## Prototype

`UD_CC_NETLOGONENCRYPT`

## Description

Whether NQ Client supports Netlogon secure channel encryption

## Notes

When this is defined, NQ encrypts the traffic over secure Netlogon channel when communicating with Active Directory. Enabled by default, can be disabled for debug purposes.



### 7.3.2 Performance Tuning

These parameters affect NQ performance and footprint.

### 7.3.2.1 UD\_NQ\_HOSTNAMESIZE

Prototype

UD\_NQ\_HOSTNAMESIZE

Description

Maximum length of a fully-qualified host name

Notes

DNS specification states that a fully qualified host name may be as long as 256 characters. Since NQ allocates storage for host name it allows reducing the footprint by decreasing this value. NQ uses approximately the following amount of memory for storing host names:

$UD\_NQ\_HOSTNAMESIZE * N$

where N is:

$UD\_CC\_MOUNTTABSIZ + UD\_CC\_SEARCHHANDLETABSIZ + 6$

While decreasing this value significantly lowers the footprint, it may contradict to certain network environments. The recommended default value is 256 for full compliance with DNS specifications.

### 7.3.2.2 UD\_NQ\_MAXDNSSERVERS

Prototype

UD\_NQ\_MAXDNSSERVERS

Description

Number of supported DNS servers

Notes

NQ uses DNS servers for name resolution. It may use a list of DNS servers to perform fallback from left to right. This parameter defines maximum number of DNS servers in the list. See also 7.2.2.4

### 7.3.2.3 UD\_NQ\_MAXWINSSERVERS

Prototype

UD\_NQ\_MAXDNSSERVERS

Description

Number of supported WINS servers

Notes

NQ uses WINS servers for name resolution. It may use a list of DNS servers to perform fallback from left to right. This parameter defines maximum number of WINS servers in the list

#### 7.3.2.4 UD\_NS\_NUMDDCHANNELS

Prototype

UD\_NS\_NUMDDCHANNELS

Description

Number of internal sockets from applications to NetBIOS Datagram Service

Notes

NQ uses internal (loop-back) sockets for communications between NQ Client from one side and NetBIOS on other side.

The maximum number for this parameter should not exceed: <number of concurrent tasks using NQ Client> + 1. The default number of two is sufficient in most cases.

NQ requires 10 bytes per internal socket

### 7.3.2.5 UD\_NS\_NUMNDCHANNELS

Prototype

UD\_NS\_NUMNDCHANNELS

Description

Number of internal sockets from applications to NetBIOS Naming Service

Notes

NQ uses internal (loop-back) sockets for communications between NQ Client from one side and NetBIOS on other side.

The maximum number for this parameter should not exceed: <number of concurrent tasks using NQ Client> + 1. The default number of two is sufficient in most cases.

NQ requires 10 bytes per internal socket

### 7.3.2.6 UD\_NS\_BUFFERSIZE

Prototype

UD\_NS\_BUFFERSIZE

Description

Size of data buffer

Notes

This value is the maximum length of a TCP or UDP message. Increasing this value may increase the performance. The exact performance growth depends on the underlying TCP/UDP/IP layer.

The recommended value differs for SMB1-only and SMB2 support.

When NQ supports SMB2 this value should be at least 64K bytes + headers which makes the number of 65700. This is required because Windows 7 does not appreciate server's limits and maximum buffer size in particular when it comes to bulk read and write operations.

Buffers size for SMB1 support may be any number greater than 1460 and not exceeding 1460x44. A multiple of 1460 is recommended to decrease TCP fragmentation. To achieve optimal performance, it is recommended testing on a real platform within the range of  $1460 * 4$  to  $1460 * 42$ .

Furthermore to avoid fragmentation of the CIFS/SMB buffers in TCP/IP stack the underlying TCP/IP stack should have at least 4 buffers big enough to accommodate  $UD\_NS\_BUFFERSIZE + 128$  rounded to the closest power of 2 up. For example, if UD\_NS\_BUFFERSIZE is set to  $1460 * 21$ , then there should be 4 TCP buffers of 32 KB size.

### 7.3.2.7 UD\_NS\_NUMBUFFERS

Prototype

UD\_NS\_NUMBUFFERS

Description

Number of data buffers to allocate

Notes

The following parameter is masked by default. Then the default value (of 2) is used which guarantees the minimum footprint. A value greater than 2 may be specified to enforce better performance. Specifying this parameter has no effect while UD\_NS\_ASYNCSEND (7.3.2.8) is masked. Each buffer adds UD\_NS\_BUFFERSIZE (7.3.2.6) bytes to the footprint.

Function *udAllocateBuffer* (see 7.2.2.18) should supply the required number of buffers while function *udReleaseBuffer* (see 7.2.2.19) should be able to release those buffers if necessary.



### 7.3.2.8 UD\_NS\_ASYNCSEND

Prototype

UD\_NS\_ASYNCSEND

Description

Asynchronous send feature

Notes

When this parameter is defined, send operation is asynchronous. It returns immediately while the buffer content is being sent in a background process. This feature requires OS support for asynchronous socket operations. OS-level asynchronous send should be implemented in *sySendSocketAsync()* call (6.4.1.6.31). Comment this line to use *sySendSocket()* call instead (6.4.1.6.29).

### 7.3.2.9 UD\_ND\_DAEMON\_TIMEOUT

Prototype

UD\_ND\_DAEMON\_TIMEOUT

Description

Daemon timeout resolution in seconds

Notes

This value defines NetBIOS daemon timeout resolution. The higher is this value the less is the resolution of calculating timeouts. Increasing this value may improve performance, yet name registration and name resolution behavior will be inaccurately timed.

**7.3.2.10    UD\_ND\_MAXINTERNALNAMES**

## Prototype

`UD_ND_MAXINTERNALNAMES`

## Description

Size of NQ table of internal names

## Notes

Internal names are those registered from inside NQ software. The required number may be calculated as: <number of concurrent tasks using NQ Client> + 2.

**7.3.2.11      UD\_ND\_MAXEXTERNALNAMES**

## Prototype

`UD_ND_MAXEXTERNALNAMES`

## Description

Size of NQ table of external names

## Notes

External names are those resolved by NQ. The required number can be calculated as number of different hosts that may be connected by NQ Client. Host name is found at beginning of mount path (see 7.2.2.12).

**7.3.2.12    UD\_ND\_MAXQUERYREQUESTS**

## Prototype

`UD_ND_MAXQUERYREQUESTS`

## Description

Maximum number of concurrent query requests

## Notes

NQ allows to several tasks to connect to the same host simultaneously over NQ Client. This may require concurrent queries for the same name. This number may be calculated as the maximum number of concurrent tasks using NQ Client. However, a user may build applications in such a way that this situation will never happen. Then this number may be set to one.

**7.3.2.13      UD\_ND\_REGISTRATIONCOUNT**

## Prototype

`UD_ND_REGISTRATIONCOUNT`

## Description

Number of retries for name registration

## Notes

The default value is that required by RFC1002 of NetBIOS. Increasing this value may improve NQ Client start up time and NQ Client connection time.

**7.3.2.14 UD\_FS\_FILENAMELEN**

Prototype

UD\_FS\_FILENAMELEN

Description

Maximum file name length

Notes

Maximum bytes (including trailing zero) in name of a file relative to its share

**7.3.2.15      UD\_FS\_FILENAMECOMPONENTLEN**

Prototype

UD\_FS\_FILENAMECOMPONENTLEN

Description

Maximum name of a file or a directory in a full path

Notes



**7.3.2.16      UD\_CC\_READBUFFERSIZE**

## Prototype

`UD_CC_READBUFFERSIZE`

## Description

Size of the NQ Client read cash

## Notes

NQ Client driver is capable of buffered read. To disable buffered read set this value to zero.

Buffered read is available only when using NQ Client over the "*net*" driver.

This parameter is only valid for VxWorks platforms. On other platforms it is ignored.

**7.3.2.17 UD\_CC\_MOUNTTABSIZ**

## Prototype

`UD_CC_MOUNTTABSIZ`

## Description

Size of the NQ mount table

## Notes

This parameter is deprecated and is stated here for backwards compatibility only. Since NQ code does not use it anymore, this parameter may be removed.

**7.3.2.18 UD\_CC\_SESSIONTABSIZ**

## Prototype

`UD_CC_SESSIONTABSIZ`

## Description

Size of the NQ session table

## Notes

This parameter is deprecated and is stated here for backwards compatibility only. Since NQ code does not use it anymore, this parameter may be removed.

**7.3.2.19 UD\_CC\_SEARCHHANDLETABSIZ**

## Prototype

`UD_CC_SEARCHHANDLETABSIZ`

## Description

Maximum number of concurrent search operations

## Notes

This parameter is deprecated and is stated here for backwards compatibility only. Since NQ code does not use it anymore, this parameter may be removed.

**7.3.2.20      UD\_CC\_FILEHANDLETABSIZE**

## Prototype

`UD_CC_FILEHANDLETABSIZE`

## Description

Maximum number of concurrently opened files

## Notes

This parameter is deprecated and is stated here for backwards compatibility only. Since NQ code does not use it anymore, this parameter may be removed.

**7.3.2.21    UD\_CC\_CONNECTIONTABSIZE**

## Prototype

`UD_CC_CONNECTIONTABSIZE`

## Description

Maximum number of NQ Client connections

## Notes

This parameter is deprecated and is stated here for backwards compatibility only. Since NQ code does not use it anymore, this parameter may be removed.

**7.3.2.22    UD\_CC\_CLIENTRESPONSETIMEOUT**

Prototype

UD\_CC\_CLIENTRESPONSETIMEOUT

Description

NQ Client timeout

Notes

Number of seconds the NQ client waits for response before fail the operation

**7.3.2.23      UD\_NQ\_TCPSOCKETSENDDTIMEOUT**

## Prototype

`UD_NQ_TCPSOCKETSENDDTIMEOUT`

## Description

TCP sockets send timeout value in seconds

## Notes

NQ uses blocking sockets, so defining timeout can significantly decrease blocking time in case of unavailable network.



**7.3.2.24 UD\_NQ\_TCPSOCKETRECVTIMEOUT**

## Prototype

`UD_NQ_TCPSOCKETRECVTIMEOUT`

## Description

TCP sockets receive timeout value in seconds

## Notes

This timeout relates to reading SMB packets only.  
NQ uses blocking sockets, so defining timeout can significantly decrease blocking time in case of unavailable network.  
Timeout value must not be less than 1.

**7.3.2.25      UD\_NQ\_SOCKETCONNECTTIMEOUT**

## Prototype

`UD_NQ_SOCKETCONNECTTIMEOUT`

## Description

Sockets connect timeout value in seconds

## Notes

NQ uses blocking sockets, so defining timeout can significantly decrease blocking time in case of unavailable network.

## 8 Implementation Manual

Some projects may require a solution that does not fit into the default implementation (as described in 2.2 and [1]). In this case a user may need to rewrite one or more of UD functions or redefine one or more of compilation parameters (see 7.2.2.22). This section describes in-depth the most crucial of the above.

### 8.1 Buffer Allocation

NQ uses a pool of buffers. All buffers are the same size as defined by `UD_NS_BUFFERSIZE` parameter (see 7.3.2.6). NQ pre-allocates buffers on start-up by repeatedly calling function `udAllocateBuffer()` (see 7.2.2.18) `UD_NS_NUMBUFFERS` times (see 7.3.2.7).

The same buffer is used for composing a protocol (CIFS/SMB, NetBIOS messages and for file input/output). For this reason NQ allocates buffers in the project (integration level). We suggest the following methods of buffer allocation:

- From a static array of buffers.
- As a memory block from dynamic memory (the heap)
- As a block from in special memory space (e.g., - non-cacheable memory).

Besides, a buffer may be required to be specifically aligned. The first method is the most recommended. First two methods are illustrated by the following code:

```
/* Statically allocated buffers buffers in the user space. */

unsigned char staticBuffers[UD_NS_NUMBUFFERS*UD_NS_BUFFERSIZE];

unsigned char* udfAllocateBuffer(
    int idx,
    unsigned int numBufs,
    unsigned int bufferSize
)
{
    return staticBuffers + idx*(bufferSize);
}

/* Buffers are allocated from dynamic memory.. */

unsigned char* udfAllocateBuffer(
    int idx,
    unsigned int numBufs,
    unsigned int bufferSize
)
{
    return malloc(bufferSize);
}
```

### 8.2 Choosing Buffer Size

NQ Client choose its buffer size as a minimum between the SMB Server Max Buffer Size and by `UD_NS_BUFFERSIZE` parameter (see 7.3.2.6).

This parameter affects NQ Client performance since NQ performance depends on the length of data that the server may handle on upload or download in one SMB message. The relevant SMB commands are READ (READ\_ANDX) and WRITE (WRITE\_ANDX).

To adjust this parameter to the optimum value the following factors should be considered:

- This discussed number is a maximum size of a CIFS/SMB message. Such message may be transferred over the network as one or more network packets. Underneath protocols (NetBIOS, TCP, IP) adds another 40 bytes of data to each network packet. To avoid unnecessary fragmentation a CIFS/SMB message should fit into a whole number of network packets. Maximum network packet over Ethernet is 1500 bytes (MTU). Thus, buffer size should be  $N \times 1460$ , where N is a whole number. The default is 1460.
- Generally speaking the greater is buffer size the better is performance. Very long buffers, however, may cause fragmentation on the TCP level. When a CIFS/SMB message is queued for sending, TCP copies it into an internal buffer. This buffer is not guaranteed to accommodate the entire message. A long message may be fragmented and the resulting performance may be even worse then with a smaller buffer. The solution may be either 1) enforcing the TCP stack to use longer buffers or 2) avoiding increasing the NQ buffer size (UD\_NS\_BUFFERSIZE) above the value, supported by the TCP stack.
- Bigger buffers require more memory.

### **8.3 64-bit File Offsets**

This section is mostly applicable to Linux platforms as well as to platforms with Posix file system layer.

The Linux sample implementation assumes that the underlying file system has 32-bit file offsets as supported by basic Posix. The same sample implementation may also support 64-bit file offsets. For this purpose, uncomment the line saying:

```
#define LONG_FILES_SUPPORT
```