

YNQ Datasheet

YNQ 1.5.0

Table of Contents

1	INTRODUCTION.....	3
2	YNQ DESCRIPTION	4
3	YNQ SPECIFICATIONS.....	5
4	INTERFACE WITH EMBEDDED/MOBILE REAL-TIME OS.....	7

Table of Figures

FIGURE 1 – YNQ OS INTERFACE	7
-----------------------------------	---

1 Introduction

YNQ is a highly portable embedded/mobile implementation of SMB/CIFS protocol also known as SMB/SMB2 or Microsoft Windows File Sharing. This implementation may be integrated in virtually any embedded/mobile application running on any operating system. YNQ features SMB/CIFS Server and SMB/CIFS Client capabilities.

YNQ Server seamlessly exposes the target device and its local files and printers to the network. By announcing itself on the network YNQ Server makes the target device visible under "My Network Places" category of any MS Windows™-based computers as if it was a regular Windows PC.

YNQ Client allows a device to access remotely shared files and printers. With NQ Client an application can browse the network by enumerating domains/workgroups on the network, hosts on the domains/workgroups and shares on hosts.

YNQ is portable, compact and high-performance solution for embedded/mobile systems.

2 YNQ Description

The SMB/CIFS file and printer sharing protocol is a native protocol for sharing files and printers between MS Windows based computers. YNQ implements the latest SMB/CIFS dialects. The first dialect is NT LM 0.12 of SMB/CIFS as described in "[Common Internet File System \(CIFS\) Technical Reference](#)" published by [SNIA](#). The second dialect is SMB 2.002 also known as SMB2. This is the preferred dialect by Microsoft Windows versions starting from Windows Vista. The third dialect is SMB 3.0.0 and its main feature is encryption. The latest dialect is SMB 3.1.1 that using GCM algorithm for better encryption performance.

YNQ allows SMB/CIFS to be carried by any of three transports: 1) NetBIOS, 2) TCP/IP(v4), also known as "naked SMB/CIFS" and, 3) TCP/IPV6.

In the first case YNQ uses an implementation of the NetBIOS over TCP/IP(v4) transport as the underlying communications layer. This implementation is fully RFC1001 and RFC1002 compliant.

YNQ contains three major modules:

- *NetBIOS Daemon* (NBD) provides NetBIOS transport and services for other modules. YNQ uses it for registering host name(s) on the network, resolving remote host names, accepting incoming sessions and broadcasting announcements and requests.
- *CIFS Server* exposes local files to the network. It reads a list of shares and maps each share on a local directory sub-tree, making them visible and accessible on the network.
- *CIFS Client* installs a network device. After mounting one or more remote file systems onto this device it provides access to the files on those file systems. An application may either access files over the installed network drive or use direct API for more comprehensive operations. Browsing capabilities of NQ Client allow step-by-step discovering of the network configuration.

YNQ uses a library of a thin system-abstraction layer. Most of system-abstraction calls are Posix-compliant. The task of porting YNQ to yet another operating system results in modifying this very small peace of code.

YNQ runs two tasks (processes):

- NetBIOS daemon.
- YNQ Server.
 - There is a configuration that can provide the NetBIOS daemon and the YNQ Server in one task (process).

YNQ Client is a library and it runs in the framework of user application.

3 YNQ Specifications

- CIFS implementation complies with SMB/CIFS specification as defined in "Common Internet File System (CIFS) Technical Reference" by SNIA.
- NetBIOS implementation complies fully with RFC 1001 and RFC 1002 and provides NetBIOS name resolution through B or H nodes.
- Highly portable ANSI C compliant code ensures ease of porting to any standards-based operating system.
- Support for NTLM 0.12 dialect (the highest SMB dialect).
- Support for SMB2.002 to SMB3.1.1 dialect.
- Recognize long file name format.
- Supports ANSI (various code pages) and Unicode character sets.
- Supports SMB/CIFS over NetBIOS, SMB/CIFS over IPv4 and SMB/CIFS over IPv6.
- Ported onto many operating systems, including, but not limited to: VxWorks, Linux, FreeBSD, Lynx, QNX, INTEGRITY, Windows CE, ITRON, SE-OS, iOS, Nucleus and ThreadX.
- Hardware platform independent.
- Tested with the following clients and servers:
Windows XP/Vista/Windows7/ Windows8/ Windows10/Windows Server 2003/2008/2012/2016/2019, Leopard, Snow Leopard, Lion, Mountain Lion, Mavericks, Yosemite, El-Capitan, Sierra, High Sierra, Mojave, Samba2, Samba3 and Samba4. YNQ Client tested also with Microsoft Azure.
- Code Size: 150 – 400 KB (depending on Operating System, processor and configuration).
- Configurable resources. All resource limit parameters have optimal default values.
- Flexible configuration. Most parameters may be changed in run-time, including network interfaces, WINS, any number of DNS servers, etc.
- Support for interfaces with arbitrary number of IPv4 and IPv6 addresses.
- YNQ Server
 - User Level Security. LM, NTLM, LMv2, NTLMv2 and NTLMSSP authentication. Supports both local and domain users using pass-through authentication in the latter case.
 - Full support of abstract file system equivalent to POSIX.
 - Partial support of NTFS extensions (as far as supported by the underlying file system).
 - Domain authentication passthrough, allows NQ to verify the incoming credentials using either Windows Domain or Samba Domain (AD).
 - Supports SMB/CIFS message signing.

- Supports SMB/CIFS ENCRYPTION.
- Notification of changes in a directory content.
- Security descriptors support for local and domain users.
- Share-level ACLs supported.
- Remote user and session management supported.
- Printing functionality supported.
- Configurable resource limits:
 - Maximum number of concurrent clients connected.
 - Maximum number of concurrently accessed files or directories.
 - Maximum number of exposed shares.
- YNQ Client
 - User or Shared level Security. LM, NTLM, LMv2, NTLMv2, NTLMSSP and Kerberos authentication.
 - Supports SMB/CIFS message signing.
 - Supports SMB/CIFS encryption.
 - Network drive. This feature allows transparent access to remote files by means of BSD file calls, currently available on 1) VxWorks and 2) Linux over FUSE.
 - DFS (distributed filesystem) support.
 - Domain membership allows NQ based computer to become a member of either Windows Domain or Samba Domain.
 - NQ Client API for more comprehensive access.
 - Network browsing capabilities in terms of: domains, hosts, shares.
 - LDAP client API (requires external LDAP client).
 - Configurable resource limits:
 - Maximum number of mounted remote files systems.
 - Maximum number of opened remote files or directories.
 - Maximum number of user sessions.

4 Interface with Embedded/Mobile Real-Time OS

Figure 1 shows the interface between YNQ Client/Server and the embedded/mobile Real Time Operating System.

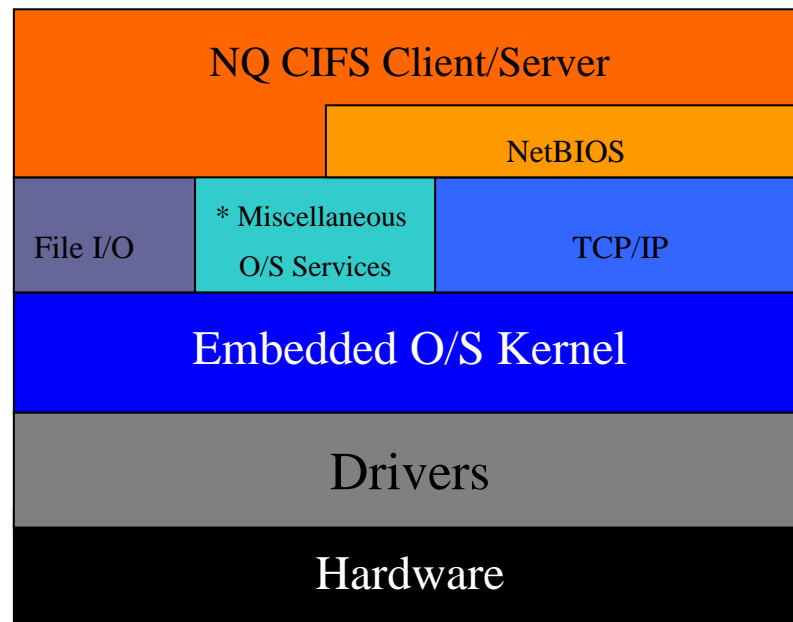


Figure 1 – NQCIFS OS interface

* Miscellaneous O/S Services: Miscellaneous OS system calls like timer, process, memory, etc.

YNQ uses the concept of system-abstraction layer. This thin layer provides the rest of NQ with system-independent API for system-dependent services. Most of system-abstraction calls may be directly mapped on ANSI/POSIX/BSD calls. For instance, function *syCreateSocket()* of this layer simply delegates its call to BSD *socket()* function.

Porting to yet another operating system requires from the underlying system to support the following library calls or their equivalents (based on the VxWorks version):

<u>Socket:</u>	<code>socket()</code>	<code>toupper()</code>	<code>mktime()</code>
<code>accept()</code>	<code>setsockopt()</code>	<code>tolower()</code>	
<code>bind()</code>			<u>File I/O:</u>
<code>connect()</code>	<u>String:</u>	<u>Memory:</u>	<code>close()</code>
<code>gethostname()</code>	<code>sprintf()</code>	<code>memcpy()</code>	<code>closedir()</code>
<code>getsockname()</code>	<code>sscanf()</code>	<code>memset()</code>	<code>creat()</code>
<code>inet_addr()</code>	<code>strcat()</code>		<code>fstat()</code>
<code>ioctl()</code>	<code>strchr()</code>	<u>Process, task:</u>	<code>ftruncate()</code>
<code>listen()</code>	<code>strcmp()</code>	<code>taskSpawn()</code>	<code>lseek()</code>
<code>recv()</code>	<code>strcpy()</code>	<code>taskIdSelf()</code>	<code>mkdir()</code>
<code>recvfrom()</code>	<code>strlen()</code>		<code>open()</code>
<code>select()</code>	<code>strlwr()</code>	<u>Time:</u>	<code>opendir()</code>
<code>send()</code>	<code>strncmp()</code>	<code>time()</code>	<code>read()</code>
<code>sendto()</code>	<code>strncpy()</code>	<code>utime()</code>	<code>readdir()</code>
<code>setsockopt()</code>	<code>strrchr()</code>	<code>gmtime_r()</code>	<code>rename()</code>
<code>shutdown()</code>	<code>strtok()</code>	<code>localtime_r()</code>	<code>stat()</code>

statfs()
unlink()
write()
remove()

Misc:
srand()

rand()
splice()
sendfile()

YNQ Client/Server requires minimal time and effort when porting it to any Operating System that directly answers the above requirements. Porting to such OS's will only involve reconfiguring API mappings in one common system-specific file to point to the new OS APIs with virtually no adaptation efforts.

Porting to less compliant OSes will require adaptation efforts at the interface between the NetBIOS and the TCP/IP layer of the OS, and also at the File I/O interface.

Figure 1 shows the way YNQ Client/Server interfaces with embedded/mobile operating systems that exhibit less than full compliance with POSIX and Berkeley Sockets.