



## **NetX Duo DNS (Domain Name System) Client**

# **User Guide**

**Express Logic, Inc.**

858.613.6640  
Toll Free 888.THREADX  
FAX 858.521.4259

[www.expresslogic.com](http://www.expresslogic.com)

**©2002-2017 by Express Logic, Inc.**

All rights reserved. This document and the associated NetX software are the sole property of Express Logic, Inc. Each contains proprietary information of Express Logic, Inc. Reproduction or duplication by any means of any portion of this document without the prior written consent of Express Logic, Inc. is expressly forbidden. Express Logic, Inc. reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of NetX. The information in this document has been carefully checked for accuracy; however, Express Logic, Inc. makes no warranty pertaining to the correctness of this document.

**Trademarks**

NetX, NetX Duo, Piconet, and UDP Fast Path are trademarks of Express Logic, Inc. ThreadX is a registered trademark of Express Logic, Inc.

All other product and company names are trademarks or registered trademarks of their respective holders.

**Warranty Limitations**

Express Logic, Inc. makes no warranty of any kind that the NetX products will meet the USER's requirements, or will operate in the manner specified by the USER, or that the operation of the NetX products will operate uninterrupted or error free, or that any defects that may exist in the NetX products will be corrected after the warranty period. Express Logic, Inc. makes no warranties of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with respect to the NetX products. No oral or written information or advice given by Express Logic, Inc., its dealers, distributors, agents, or employees shall create any other warranty or in any way increase the scope of this warranty, and licensee may not rely on any such information or advice.

Part Number: 000-1051

Revision 5.10

# Contents

---

|  |    |
|--|----|
| Chapter 1 Introduction to the NetX Duo DNS Client .....    | 4  |
| DNS Client Setup .....                                     | 4  |
| DNS Messages .....   | 5  |
| Extended DNS Resource Record Types .....                   | 6  |
| DNS Cache .....  | 8  |
| DNS Client Limitations .....                               | 8  |
| DNS RFCs.....  | 8  |
| Chapter 2 Installation and Use of NetX Duo DNS Client..... | 10 |
| Product Distribution .....                                 | 10 |
| DNS Client Installation .....                              | 10 |
| Using the DNS Client .....                                 | 10 |
| Small Example System for NetX Duo DNS Client .....         | 11 |
| Configuration Options.....                                 | 25 |
| Chapter 3 Description of DNS Client Services .....         | 28 |
| nx_dns_authority_zone_start_get .....                      | 31 |
| nx_dns_cache_initialize .....                              | 35 |
| nx_dns_cache_notify_clear .....                            | 36 |
| nx_dns_cache_notify_set .....                              | 37 |
| nx_dns_cname_get .....                                     | 38 |
| nx_dns_create .....  | 40 |
| nx_dns_delete .....  | 41 |
| nx_dns_domain_name_server_get .....                        | 42 |
| nx_dns_domain_mail_exchange_get .....                      | 45 |
| nx_dns_domain_service_get .....                            | 48 |
| nx_dns_get_serverlist_size .....                           | 51 |
| nx_dns_info_by_name_get .....                              | 52 |
| nx_dns_ipv4_address_by_name_get .....                      | 54 |
| nxd_dns_ipv6_address_by_name_get .....                     | 56 |
| nx_dns_host_by_address_get .....                           | 58 |
| nxd_dns_host_by_address_get .....                          | 60 |
| nx_dns_host_by_name_get .....                              | 62 |
| nxd_dns_host_by_name_get .....                             | 64 |
| nx_dns_host_text_get .....                                 | 67 |
| nx_dns_packet_pool_set .....                               | 69 |
| nx_dns_server_add .....                                    | 71 |
| nxd_dns_server_add .....                                   | 72 |
| nx_dns_server_get .....                                    | 74 |
| nxd_dns_server_get .....                                   | 76 |
| nx_dns_server_remove .....                                 | 78 |
| nxd_dns_server_remove .....                                | 80 |
| nx_dns_server_remove_all .....                             | 82 |

# Chapter 1

## Introduction to the NetX Duo DNS Client

The DNS provides a distributed database that contains mapping between domain names and physical IP addresses. The database is referred to as *distributed* because there is no single entity on the Internet that contains the complete mapping. An entity that maintains a portion of the mapping is called a DNS Server. The Internet is composed of numerous DNS Servers, each of which contains a subset of the database. DNS Servers also respond to DNS Client requests for domain name mapping information, only if the server has the requested mapping.

The DNS Client protocol for NetX Duo provides the application with services to request mapping information from one or more DNS Servers.

### DNS Client Setup

In order to function properly, the DNS Client package requires that a NetX Duo IP instance has already been created.

After creating the DNS Client, the application must add one or more DNS servers to the server list maintained by the DNS Client. To add DNS servers, the application uses the `nxd_dns_server_add` service. The NetX Duo DNS Client service `nx_dns_server_add` can also be used to add servers. However it only accepts IPv4 addresses and it is recommended that developers use the `nxd_dns_server_add` service instead.

If the `NX_DNS_IP_GATEWAY_SERVER` option is enabled, and the IP instance gateway address is non zero, the IP instance gateway is automatically added as the primary DNS server. If DNS server information is not statically known, it may also be derived through the Dynamic Host Configuration Protocol (DHCP) for NetX Duo. Please refer to the NetX Duo DHCP User Guide for more information.

The DNS Client requires a packet pool for transmitting DNS messages. By default, the DNS Client creates this packet pool when the `nx_dns_create` service is called. The configuration options `NX_DNS_PACKET_PAYLOAD_UNALIGNED` and `NX_DNS_PACKET_POOL_SIZE` allow the application to determine the packet payload and packet pool size (e.g. number of packets) of this packet pool respectively. These options are described in section “Configuration Options” in Chapter Two.

An alternative to the DNS Client creating its own packet pool is for the application to create the packet pool and set it as the DNS Client's packet pool using the *nx\_dns\_packet\_pool\_set* service. To do so, the `NX_DNS_CLIENT_USER_CREATE_PACKET_POOL` option must be defined. This option also requires a previously created packet pool using *nx\_packet\_pool\_create* as the packet pool pointer input to *nx\_dns\_packet\_pool\_set*. When the DNS Client instance is deleted, the application is responsible for deleting the DNS Client packet pool if `NX_DNS_CLIENT_USER_CREATE_PACKET_POOL` is enabled if it is no longer needed.

**Note:** For applications choosing to provide its own packet pool using the `NX_DNS_CLIENT_USER_CREATE_PACKET_POOL` option, the packet size needs to be able to hold the DNS maximum message size (512 bytes) plus rooms for UDP header, IPv4 or IPv6 header, and the MAC header.

## DNS Messages

The DNS has a very simple mechanism for obtaining mapping between host names and IP addresses. To obtain a mapping, the DNS Client prepares a DNS query message containing the name or the IP address that needs to be resolved. The message is then sent to the first DNS server in the server list. If the server has such a mapping, it replies to the DNS Client using a DNS response message that contains the requested mapping information. If the server does not respond, the DNS Client queries the next server on its list until all its DNS servers have been queried. If no response from all its DNS servers is received, the DNS Client has retry logic to retransmit the DNS message. On resending a DNS query, the retransmission timeout is doubled. This process continues until the maximum transmission timeout (defined as `NX_DNS_MAX_RETRANS_TIMEOUT` in *nxd\_dns.h*) is reached or until a successful response is received from that server is obtained.

NetX Duo DNS Client can perform both IPv6 address lookups (type AAAA) and IPv4 address lookups (type A) by specifying the version of the IP address in the *nxd\_dns\_host\_by\_name\_get* call. The DNS Client can perform reverse lookups of IP addresses (type PTR queries) to obtain web host names using *nxd\_dns\_host\_by\_address\_get*. The NetX Duo DNS Client still supports the *nx\_dns\_host\_by\_name\_get* and *nx\_dns\_host\_by\_address\_get* which are the equivalent services but which are limited to IPv4 network communication. However, developers are encouraged to port existing DNS Client applications to the *nxd\_dns\_host\_by\_name\_get* and *nxd\_dns\_host\_by\_address\_get* services.

DNS messaging utilizes the UDP protocol to send requests and field responses. A DNS Server listens on port number 53 for queries from clients. Therefore UDP services must be enabled in NetX Duo using the ***nx\_udp\_enable*** service on a previously created IP instance (***nx\_ip\_create***).

At this point, the DNS Client is ready to accept requests from the application and send out DNS queries.

## Extended DNS Resource Record Types

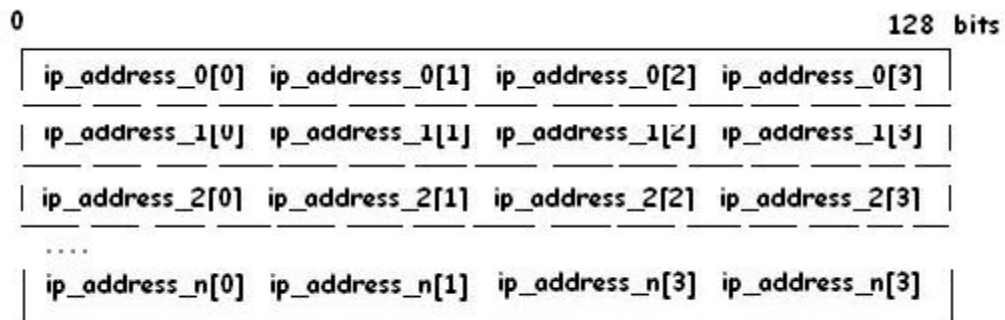
If `NX_DNS_ENABLE_EXTENDED_RR_TYPES` is enabled, NetX Duo DNS Client also supports the following record type queries:

|       |   |
|-------|---|
| CNAME | contains the canonical name for an alias                  |
| TXT   | contains a text string                                    |
| NS    | contains an authoritative name server                     |
| SOA   | contains the start of a zone of authority                 |
| MX    | used for mail exchange                                    |
| SRV   | contains information on the service offered by the domain |

With the exception of CNAME and TXT record types, the application must supply a 4-byte aligned buffer to receive the DNS data record.

In NetX Duo DNS Client, record data is stored in such a way to make most efficient use of buffer space.

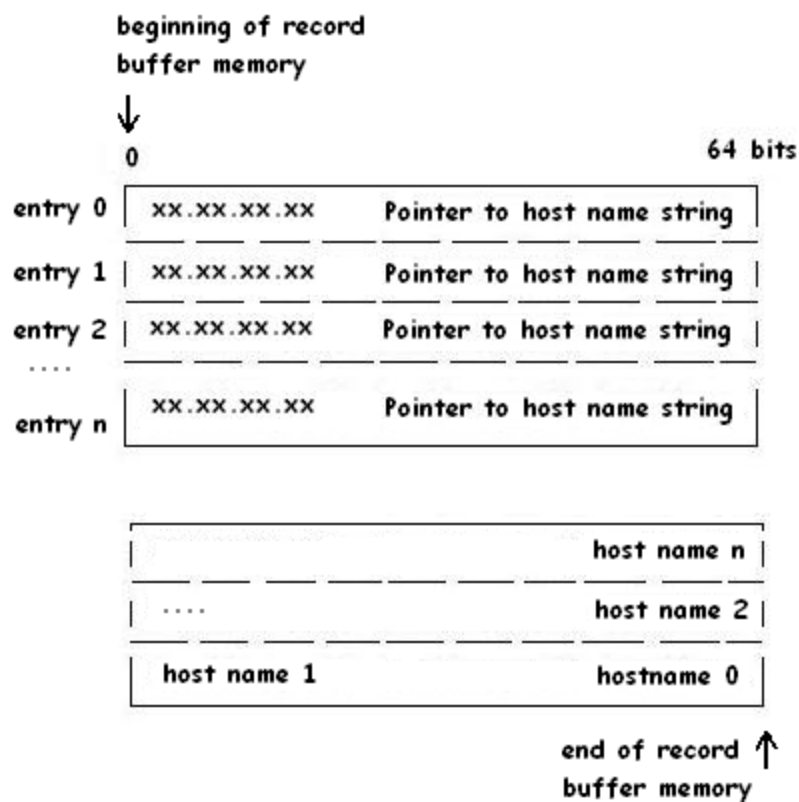
An example of a record buffer of fixed length (type AAAA record) is shown below:



For those queries whose record types have variable data length, such as NS records whose host names are of variable length, NetX Duo DNS Client saves the data as follows. The buffer supplied in the DNS Client query is

organized into an area of fixed length data and an area of unstructured memory. The top of the memory buffer is organized into 4-byte aligned record entries. Each record entry contains the IP address and a pointer to the variable length data for that IP address. The variable length data for each IP address are stored in the unstructured area memory starting at the end of the memory buffer. The variable length data for each successive record entry is saved in the next area memory adjacent to the previous record entries variable data. Hence, the variable data 'grows' towards the structured area of memory containing the record entries until there is insufficient memory to store another record entry and variable data.

This is shown in the figure below:



The example of the DNS domain name (NS) data storage is shown above.

NetX Duo DNS Client queries using the record storage format return the number of records saved to the record buffer. This information enables the application to extract NS records from the record buffer.

An example of a DNS Client query that stores variable length DNS data using this record storage format is shown below:

```
UINT _nx_dns_domain_name_server_get(NX_DNS *dns_ptr,
                                     UCHAR *host_name, VOID *record_buffer,
                                     UINT buffer_size, UINT *record_count,
                                     ULONG wait_option)
```

More details are available in Chapter 3, “Description of DNS Client Services”.

## DNS Cache

If `NX_DNS_CACHE_ENABLE` is enabled, NetX Duo DNS Client supports the DNS Cache feature. After creating the DNS Client, the application can call the API `nx_dns_cache_initialize()` to set the special DNS Cache. If enable DNS Cache feature, DNS Client will find the available answer from DNS Cache before starts to send DNS query, if find the available answer, directly return the answer to application, otherwise DNS Client sends out query message to DNS server and waits for the reply. When DNS Client gets the response message and there is free cache available, DNS Client returns the answer to the application and also adds the answer as resource record into DNS cache.

Each answer a data structure `NX_DNS_RR` (Resource Record) in the cache. Strings (resource record name and data) in Records are variable length, therefore are not stored in the `NX_DNS_RR` structure. The Record contains pointers to the actual memory location where the strings are stored. The string table and the Records share the cache. Records are stored from the beginning of the cache, and grow towards the end of the cache. The string table starts from the end of the cache and grows towards the beginning of the cache. Each string in the string table has a length field and a counter field. When a string is added to the string table, if the same string is already present in the table, the counter value is incremented and no memory is allocated for the string. The cache is considered full if no more resource records or new strings can be added to the cache.

## DNS Client Limitations

The DNS Client supports one DNS request at a time. Threads attempting to make another DNS request are temporarily blocked until the previous DNS request is complete.

The NetX Duo DNS Client does not use data from authoritative answers to forward additional DNS queries to other DNS servers.

## DNS RFCs

NetX Duo DNS is compliant with the following RFCs:



RFC1034 DOMAIN NAMES - CONCEPTS AND FACILITIES  
RFC1035 DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION  
RFC1480 The US Domain  
RFC 2782 A DNS RR for specifying the location of services (DNS SRV)  
RFC 3596 DNS Extensions to Support IP Version 6

## Chapter 2

# Installation and Use of NetX Duo DNS Client

This chapter contains a description of various issues related to installation, setup, and usage of the NetX Duo DNS Client.

## Product Distribution

NetX Duo DNS Client is shipped on a single CD-ROM compatible disk. The package includes two source files and a PDF file that contains this document, as follows:

|                                 |  |
|---------------------------------|--|
| <b><code>nxd_dns.h</code></b>   | Header file for NetX Duo DNS Client    |
| <b><code>nxd_dns.c</code></b>   | C Source file for NetX Duo DNS Client  |
| <b><code>nxd_dns.pdf</code></b> | PDF description of NetX Duo DNS Client |

## DNS Client Installation

To use NetX Duo DNS Client, copy the source code files *nxd\_dns.c* and *nxd\_dns.h* to the same directory where NetX Duo is installed. For example, if NetX Duo is installed in the directory “\threadx\arm7\green” then the *nxd\_dns.h* and *nxd\_dns.c* files should be copied into this directory.

## Using the DNS Client

Using NetX Duo DNS Client is easy. Basically, the application code must include *nxd\_dns.h* after it includes *tx\_api.h* and *nx\_api.h*, in order to use ThreadX and NetX Duo, respectively. Once *nxd\_dns.h* is included, the application code is then able to make the DNS function calls specified later in this guide. The application must also add *nxd\_dns.c* to the build process. This file must be compiled in the same manner as other application files and its object form must be linked along with the files of the application. This is all that is required to use NetX Duo DNS.

Note that since DNS utilizes NetX Duo UDP services, UDP must be enabled with the *nx\_udp\_enable* call prior to using DNS.

## Small Example System for NetX Duo DNS Client

NetX Duo DNS Client is compatible with existing NetX DNS applications. The list of legacy services and their NetX Duo equivalent is shown below:

### NetX DNS API service (IPv4 only)

`nx_dns_host_by_name_get`  
`nx_dns_host_by_address_get`  
`nx_dns_server_get`  
`nx_dns_server_add`  
`nx_dns_server_remove`

### NetX Duo DNS API service (IPv4 and IPv6 supported)

`nxd_dns_host_by_name_get`  
`nxd_dns_host_by_address_get`  
`nxd_dns_server_get`  
`nxd_dns_server_add`  
`nxd_dns_server_remove`

See the description of NetX Duo DNS Client API services in Chapter 3 for more details.

In the example DNS application program provided in this section, *nxd\_dns.h* is included at line 6. `NX_DNS_CLIENT_USER_CREATE_PACKET_POOL`, which allows the DNS Client application to create the packet pool for the DNS Client, is declared on lines 24-26. This packet pool is used for allocating packets for sending DNS messages. If `NX_DNS_CLIENT_USER_CREATE_PACKET_POOL` is defined, a packet pool is created in lines 78-98. If this option is not enabled, the DNS Client creates its own packet pool as per the packet payload and pool size set by configuration parameters in *nxd\_dns.h* and described elsewhere in this chapter.

Another packet pool is created in lines 100-112 for the Client IP instance which is used for internal NetX Duo operations. Next the IP instance is created using the *nx\_ip\_create* call in line 115. It is possible for the IP task and the DNS Client to share the same packet pool, but since the DNS Client typically sends out larger messages than the control packets sent by the IP task, using separate packet pools makes more efficient use of memory.

ARP and UDP (which is used by IPv4 networks) are enabled in lines 129 and 141 respectively.

Note this demo uses the 'ram' driver declared on line 44 and used in the *nx\_ip\_create* call. This ram driver is distributed with the NetX Duo source code. To actually run the DNS Client the application must supply an actual physical network driver to transmit and receive packets from the DNS server.

The Client thread entry function *thread\_client\_entry* is defined below the *tx\_application\_define* function. It initially relinquishes control to the system to allow the IP task thread to be initialized by the network driver.

It then creates the DNS Client in line 257, initializes the DNS cache on lines 267- 278, and sets the packet pool previously created to the DNS Client instance on lines 281-295. It then adds DNS server on lines 297-341.

The remainder of the example program uses the DNS Client services to make DNS queries. Host IP address lookups are performed on lines 193 and 207. The difference between these two services, *nxd\_dns\_host\_by\_name\_get* and *nx\_dns\_host\_by\_name\_get*, is that the former saves the address data in an NXD\_ADDRESS data type, while the latter saves the data in a ULONG data type. Further the latter is limited to IPv4 networks, while the former can be used with IPv6 or IPv4 networks. This is only possible if the IP instance is enabled for IPv6. See the NetX Duo User Guide for more details on enabling the IP instance for IPv6 networking.

Another service for host IP address lookups is shown on line 464, *nx\_dns\_ipv4\_address\_by\_name\_get*. This service differs from *nx\_dns\_host\_by\_name\_get* in that it returns all (or as many will fit in the supplied buffer) of the IPv4 addresses discovered for the domain name, not just the first address received in the DNS Server reply.

Similarly, the *nxd\_dns\_ipv6\_address\_by\_name\_get* service, called on line 380, returns all the IPv6 addresses discovered by the DNS Client, not just the first one.

Reverse lookups (host name from IP address) are performed on lines 606 (*nx\_dns\_host\_by\_address\_get*) and again on line 564 and 588 (*nxd\_dns\_host\_by\_address\_get*). *nx\_dns\_host\_by\_address\_get* will only work on IPv4 networks, while *nxd\_dns\_host\_by\_address\_get* will work on either IPv4 or IPv6 networks (e.g. the IP instance is enabled for IPv6 as well as IPv4 networks).

Two more services for DNS lookups, CNAME and TXT, are demonstrated on lines 627 and 649 respectively, to discover CNAME and domain name data for the input domain name. NetX Duo DNS Client as similar services for other record types, e.g. MX, NS, SRV and SOA. See Chapter 3 for detailed descriptions of all record type lookups available in NetX Duo DNS Client.

When the DNS Client is deleted on line 846, using the *nx\_dns\_delete* service, the packet pool for the DNS Client is not deleted unless the DNS Client created its own packet pool. Otherwise, it is up to the application to delete the packet pool if it has no further use for it.

```

1 /* This is a small demo of DNS Client for the high-performance NetX TCP/IP stack.
2 */
3 #include "tx_api.h"
4 #include "nx_api.h"
5 #include "nx_udp.h"
6 #include "nxd_dns.h"
7

```

```

8 #ifdef FEATURE_NX_IPV6
9 #include "nx_ipv6.h"
10 #endif
11
12 #define DEMO_STACK_SIZE 4096
13
14 #define NX_PACKET_PAYLOAD 1536
15 #define NX_PACKET_POOL_SIZE 30 * NX_PACKET_PAYLOAD
16 #define LOCAL_CACHE_SIZE 2048
17
18 /* Define the ThreadX and NetX object control blocks... */
19
20 NX_DNS client_dns;
21 TX_THREAD client_thread;
22 NX_IP client_ip;
23 NX_PACKET_POOL main_pool;
24 #ifdef NX_DNS_CLIENT_USER_CREATE_PACKET_POOL
25 NX_PACKET_POOL client_pool;
26 #endif
27 UCHAR local_cache[LOCAL_CACHE_SIZE];
28
29 UINT error_counter = 0;
30
31 #ifdef FEATURE_NX_IPV6
32 /* If IPv6 is enabled in NetX Duo, allow DNS Client to try using IPv6 */
33 //#define USE_IPV6
34 #endif
35
36 #define CLIENT_ADDRESS IP_ADDRESS(192,168,0,11)
37 #define DNS_SERVER_ADDRESS IP_ADDRESS(192,168,0,1)
38
39 /* Define thread prototypes. */
40
41 void thread_client_entry(ULONG thread_input);
42
43 /***** Substitute your ethernet driver entry function here *****/
44 extern VOID _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
45
46
47 /* Define main entry point. */
48
49 int main()
50 {
51     /* Enter the ThreadX kernel. */
52     tx_kernel_enter();
53 }
54
55
56
57 /* Define what the initial system looks like. */
58
59 void tx_application_define(void *first_unused_memory)
60 {
61     CHAR *pointer;
62     UINT status;
63
64
65     /* Setup the working pointer. */
66     pointer = (CHAR *) first_unused_memory;
67
68     /* Create the main thread. */
69     tx_thread_create(&client_thread, "client thread", thread_client_entry, 0,
70         pointer, DEMO_STACK_SIZE, 4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
71
72     pointer = pointer + DEMO_STACK_SIZE;
73
74     /* Initialize the NetX system. */
75     nx_system_initialize();
76
77 #ifdef NX_DNS_CLIENT_USER_CREATE_PACKET_POOL
78     /* Create the packet pool for the DNS Client to send packets.
79
80     If the DNS Client is configured for letting the host application create
81     the DNS packet pool, (see NX_DNS_CLIENT_USER_CREATE_PACKET_POOL option),
82     see nx_dns_create() for guidelines on packet payload size and pool size.
83     packet traffic for NetX processes.
84     */
85
86

```

```

87     status = nx_packet_pool_create(&client_pool, "DNS Client Packet Pool",
88     NX_DNS_PACKET_PAYLOAD, pointer, NX_DNS_PACKET_POOL_SIZE);
89     pointer = pointer + NX_DNS_PACKET_POOL_SIZE;
90     /* Check for pool creation error. */
91     if (status)
92     {
93         error_counter++;
94         return;
95     }
96 #endif
97
98 /* Create the packet pool which the IP task will use to send packets. Also
99 available to the host
100 application to send packet. */
101 status = nx_packet_pool_create(&main_pool, "Main Packet Pool",
102 NX_PACKET_PAYLOAD, pointer, NX_PACKET_POOL_SIZE);
103 pointer = pointer + NX_PACKET_POOL_SIZE;
104
105 /* Check for pool creation error. */
106 if (status)
107 {
108     error_counter++;
109     return;
110 }
111
112 /* Create an IP instance for the DNS Client. */
113 status = nx_ip_create(&client_ip, "DNS Client IP Instance", CLIENT_ADDRESS,
114 0xFFFFFFFFUL,
115 &main_pool, _nx_ram_network_driver, pointer, 2048, 1);
116
117 pointer = pointer + 2048;
118
119 /* Check for IP create errors. */
120 if (status)
121 {
122     error_counter++;
123     return;
124 }
125
126 /* Enable ARP and supply ARP cache memory for the DNS Client IP. */
127 status = nx_arp_enable(&client_ip, (void *) pointer, 1024);
128 pointer = pointer + 1024;
129
130 /* Check for ARP enable errors. */
131 if (status)
132 {
133     error_counter++;
134     return;
135 }
136
137 /* Enable UDP traffic because DNS is a UDP based protocol. */
138 status = nx_udp_enable(&client_ip);
139
140 /* Check for UDP enable errors. */
141 if (status)
142 {
143     error_counter++;
144     return;
145 }
146
147 }
148
149 #define BUFFER_SIZE    200
150 #define RECORD_COUNT    10
151
152 /* Define the Client thread. */
153 void thread_client_entry(ULONG thread_input)
154 {
155     UCHAR    record_buffer[200];
156     UINT    record_count;
157     UINT    status;
158     ULONG    host_ip_address;

```

```

164 #ifdef FEATURE_NX_IPV6
165 NXD_ADDRESS    host_ipduo_address;
166 NXD_ADDRESS    test_ipduo_server_address;
167 #ifdef USE_IPV6
168 NXD_ADDRESS    client_ipv6_address;
169 NXD_ADDRESS    dns_ipv6_server_address;
170 UINT          iface_index, address_index;
171 #endif
172 #endif
173 UINT          i;
174 ULONG          *ipv4_address_ptr[RECORD_COUNT];
175 NX_DNS_IPV6_ADDRESS
176              *ipv6_address_ptr[RECORD_COUNT];
177 #ifdef NX_DNS_ENABLE_EXTENDED_RR_TYPES
178 NX_DNS_NS_ENTRY
179              *nx_dns_ns_entry_ptr[RECORD_COUNT];
180 NX_DNS_MX_ENTRY
181              *nx_dns_mx_entry_ptr[RECORD_COUNT];
182 NX_DNS_SRV_ENTRY
183              *nx_dns_srv_entry_ptr[RECORD_COUNT];
184 NX_DNS_SOA_ENTRY
185              *nx_dns_soa_entry_ptr;
186 ULONG          host_address;
187 USHORT         host_port;
188 #endif
189
190 /* Give NetX IP task a chance to get initialized . */
191 tx_thread_sleep(100);
192
193 #ifdef FEATURE_NX_IPV6
194 #ifdef USE_IPV6
195
196 /* Make the DNS Client IPv6 enabled. */
197 status = nxd_ipv6_enable(&client_ip);
198
199 /* Check for enable errors. */
200 if (status)
201 {
202
203     error_counter++;
204     return;
205 }
206 status = nxd_icmp_enable(&client_ip);
207
208 /* Check for enable errors. */
209 if (status)
210 {
211
212     error_counter++;
213     return;
214 }
215
216 client_ipv6_address.nxd_ip_address.v6[3] = 0x101;
217 client_ipv6_address.nxd_ip_address.v6[2] = 0x0;
218 client_ipv6_address.nxd_ip_address.v6[1] = 0x0000f101;
219 client_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
220 client_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6;
221
222
223 /* Set the link local address with the host MAC address. */
224 iface_index = 0;
225
226 /* This assumes we are using the primary network interface (index 0). */
227 status = nxd_ipv6_address_set(&client_ip, iface_index, NX_NULL, 10,
&address_index);
228
229 /* Check for link local address set error. */
230 if (status)
231 {
232
233     error_counter++;
234     return;
235 }
236
237 /* Set the host global IP address. We are assuming a 64
238 bit prefix here but this can be any value (< 128). */
239 status = nxd_ipv6_address_set(&client_ip, iface_index, &client_ipv6_address,
64, &address_index);
240
241 /* Check for global address set error. */
242 if (status)

```

```

243     {
244
245         error_counter++;
246         return;
247     }
248
249     /* wait while NetX Duo validates the link local and global address. */
250     tx_thread_sleep(500);
251 #endif
252 #endif
253
254     /* Create a DNS instance for the Client. Note this function will create
255     the DNS Client packet pool for creating DNS message packets intended
256     for querying its DNS server. */
257     status = nx_dns_create(&client_dns, &client_ip, (UCHAR *)"DNS Client");
258
259     /* Check for DNS create error. */
260     if (status)
261     {
262
263         error_counter++;
264         return;
265     }
266
267 #ifdef NX_DNS_CACHE_ENABLE
268     /* Initialize the cache. */
269     status = nx_dns_cache_initialize(&client_dns, local_cache, LOCAL_CACHE_SIZE);
270
271     /* Check for DNS cache error. */
272     if (status)
273     {
274
275         error_counter++;
276         return;
277     }
278 #endif
279
280     /* Is the DNS client configured for the host application to create the packet
281     pool? */
282 #ifdef NX_DNS_CLIENT_USER_CREATE_PACKET_POOL
283     /* Yes, use the packet pool created above which has appropriate payload size
284     for DNS messages. */
285     status = nx_dns_packet_pool_set(&client_dns, &client_pool);
286
287     /* Check for set DNS packet pool error. */
288     if (status)
289     {
290
291         error_counter++;
292         return;
293     }
294
295 #endif /* NX_DNS_CLIENT_USER_CREATE_PACKET_POOL */
296
297 #ifdef FEATURE_NX_IPV6
298 #ifdef USE_IPV6
299
300     /* Add an IPv6 DNS server to the DNS client. */
301     dns_ipv6_server_address.nxd_ip_address.v6[3] = 0x106;
302     dns_ipv6_server_address.nxd_ip_address.v6[2] = 0x0;
303     dns_ipv6_server_address.nxd_ip_address.v6[1] = 0x0000f101;
304     dns_ipv6_server_address.nxd_ip_address.v6[0] = 0x20010db8;
305     dns_ipv6_server_address.nxd_ip_version = NX_IP_VERSION_V6;
306
307     status = nxd_dns_server_add(&client_dns, &dns_ipv6_server_address);
308
309     /* Check for DNS add server error. */
310     if (status)
311     {
312
313         error_counter++;
314         return;
315     }
316 #else
317
318     /* Add an IPv4 server address to the Client list. */
319     status = nx_dns_server_add(&client_dns, DNS_SERVER_ADDRESS);
320
321     /* Check for DNS add server error. */
322     if (status)

```



```

323     {
324
325         error_counter++;
326         return;
327     }
328 #endif
329 #else
330
331     /* Add an IPv4 server address to the Client list. */
332     status = nx_dns_server_add(&client_dns, DNS_SERVER_ADDRESS);
333
334     /* Check for DNS add server error. */
335     if (status)
336     {
337
338         error_counter++;
339         return;
340     }
341 #endif
342
343
344
345 /*
346  *
347  *
348  *
349  *
350  *
351  *
352  *
353  *
354  *
355  *
356  *
357  *
358  *
359  *
360  *
361  *
362  *
363  *
364  *
365  *
366  *
367  *
368  *
369  *
370  *
371  *
372  *
373  *
374  *
375  *
376  *
377  *
378  *
379  *
380  *
381  *
382  *
383  *
384  *
385  *
386  *
387  *
388  *
389  *
390  *
391  *
392  *
393  *
394  *
395  *
396  *
397  *
398  *
399  *
400  *
401  *
402  *
403  *
404  *
405  *
406  *
407  *
408  *
409  *
410  *
411  *
412  *
413  *
414  *
415  *
416  *
417  *
418  *
419  *
420  *
421  *
422  *
423  *
424  *
425  *
426  *
427  *
428  *
429  *
430  *
431  *
432  *
433  *
434  *
435  *
436  *
437  *
438  *
439  *
440  *
441  *
442  *
443  *
444  *
445  *
446  *
447  *
448  *
449  *
450  *
451  *
452  *
453  *
454  *
455  *
456  *
457  *
458  *
459  *
460  *
461  *
462  *
463  *
464  *
465  *
466  *
467  *
468  *
469  *
470  *
471  *
472  *
473  *
474  *
475  *
476  *
477  *
478  *
479  *
480  *
481  *
482  *
483  *
484  *
485  *
486  *
487  *
488  *
489  *
490  *
491  *
492  *
493  *
494  *
495  *
496  *
497  *
498  *
499  *
500  *
501  *
502  *
503  *
504  *
505  *
506  *
507  *
508  *
509  *
510  *
511  *
512  *
513  *
514  *
515  *
516  *
517  *
518  *
519  *
520  *
521  *
522  *
523  *
524  *
525  *
526  *
527  *
528  *
529  *
530  *
531  *
532  *
533  *
534  *
535  *
536  *
537  *
538  *
539  *
540  *
541  *
542  *
543  *
544  *
545  *
546  *
547  *
548  *
549  *
550  *
551  *
552  *
553  *
554  *
555  *
556  *
557  *
558  *
559  *
560  *
561  *
562  *
563  *
564  *
565  *
566  *
567  *
568  *
569  *
570  *
571  *
572  *
573  *
574  *
575  *
576  *
577  *
578  *
579  *
580  *
581  *
582  *
583  *
584  *
585  *
586  *
587  *
588  *
589  *
590  *
591  *
592  *
593  *
594  *
595  *
596  *
597  *
598  *
599  *
600  *
601  *
602  *
603  *
604  *
605  *
606  *
607  *
608  *
609  *
610  *
611  *
612  *
613  *
614  *
615  *
616  *
617  *
618  *
619  *
620  *
621  *
622  *
623  *
624  *
625  *
626  *
627  *
628  *
629  *
630  *
631  *
632  *
633  *
634  *
635  *
636  *
637  *
638  *
639  *
640  *
641  *
642  *
643  *
644  *
645  *
646  *
647  *
648  *
649  *
650  *
651  *
652  *
653  *
654  *
655  *
656  *
657  *
658  *
659  *
660  *
661  *
662  *
663  *
664  *
665  *
666  *
667  *
668  *
669  *
670  *
671  *
672  *
673  *
674  *
675  *
676  *
677  *
678  *
679  *
680  *
681  *
682  *
683  *
684  *
685  *
686  *
687  *
688  *
689  *
690  *
691  *
692  *
693  *
694  *
695  *
696  *
697  *
698  *
699  *
700  *
701  *
702  *
703  *
704  *
705  *
706  *
707  *
708  *
709  *
710  *
711  *
712  *
713  *
714  *
715  *
716  *
717  *
718  *
719  *
720  *
721  *
722  *
723  *
724  *
725  *
726  *
727  *
728  *
729  *
730  *
731  *
732  *
733  *
734  *
735  *
736  *
737  *
738  *
739  *
740  *
741  *
742  *
743  *
744  *
745  *
746  *
747  *
748  *
749  *
750  *
751  *
752  *
753  *
754  *
755  *
756  *
757  *
758  *
759  *
760  *
761  *
762  *
763  *
764  *
765  *
766  *
767  *
768  *
769  *
770  *
771  *
772  *
773  *
774  *
775  *
776  *
777  *
778  *
779  *
780  *
781  *
782  *
783  *
784  *
785  *
786  *
787  *
788  *
789  *
790  *
791  *
792  *
793  *
794  *
795  *
796  *
797  *
798  *
799  *
800  *
801  *
802  *
803  *
804  *
805  *
806  *
807  *
808  *
809  *
810  *
811  *
812  *
813  *
814  *
815  *
816  *
817  *
818  *
819  *
820  *
821  *
822  *
823  *
824  *
825  *
826  *
827  *
828  *
829  *
830  *
831  *
832  *
833  *
834  *
835  *
836  *
837  *
838  *
839  *
840  *
841  *
842  *
843  *
844  *
845  *
846  *
847  *
848  *
849  *
850  *
851  *
852  *
853  *
854  *
855  *
856  *
857  *
858  *
859  *
860  *
861  *
862  *
863  *
864  *
865  *
866  *
867  *
868  *
869  *
870  *
871  *
872  *
873  *
874  *
875  *
876  *
877  *
878  *
879  *
880  *
881  *
882  *
883  *
884  *
885  *
886  *
887  *
888  *
889  *
890  *
891  *
892  *
893  *
894  *
895  *
896  *
897  *
898  *
899  *
900  *
901  *
902  *
903  *
904  *
905  *
906  *
907  *
908  *
909  *
910  *
911  *
912  *
913  *
914  *
915  *
916  *
917  *
918  *
919  *
920  *
921  *
922  *
923  *
924  *
925  *
926  *
927  *
928  *
929  *
930  *
931  *
932  *
933  *
934  *
935  *
936  *
937  *
938  *
939  *
940  *
941  *
942  *
943  *
944  *
945  *
946  *
947  *
948  *
949  *
950  *
951  *
952  *
953  *
954  *
955  *
956  *
957  *
958  *
959  *
960  *
961  *
962  *
963  *
964  *
965  *
966  *
967  *
968  *
969  *
970  *
971  *
972  *
973  *
974  *
975  *
976  *
977  *
978  *
979  *
980  *
981  *
982  *
983  *
984  *
985  *
986  *
987  *
988  *
989  *
990  *
991  *
992  *
993  *
994  *
995  *
996  *
997  *
998  *
999  *
1000  *
1001  *
1002  *
1003  *
1004  *
1005  *
1006  *
1007  *
1008  *
1009  *
1010  *
1011  *
1012  *
1013  *
1014  *
1015  *
1016  *
1017  *
1018  *
1019  *
1020  *
1021  *
1022  *
1023  *
1024  *
1025  *
1026  *
1027  *
1028  *
1029  *
1030  *
1031  *
1032  *
1033  *
1034  *
1035  *
1036  *
1037  *
1038  *
1039  *
1040  *
1041  *
1042  *
1043  *
1044  *
1045  *
1046  *
1047  *
1048  *
1049  *
1050  *
1051  *
1052  *
1053  *
1054  *
1055  *
1056  *
1057  *
1058  *
1059  *
1060  *
1061  *
1062  *
1063  *
1064  *
1065  *
1066  *
1067  *
1068  *
1069  *
1070  *
1071  *
1072  *
1073  *
1074  *
1075  *
1076  *
1077  *
1078  *
1079  *
1080  *
1081  *
1082  *
1083  *
1084  *
1085  *
1086  *
1087  *
1088  *
1089  *
1090  *
1091  *
1092  *
1093  *
1094  *
1095  *
1096  *
1097  *
1098  *
1099  *
1100  *
1101  *
1102  *
1103  *
1104  *
1105  *
1106  *
1107  *
1108  *
1109  *
1110  *
1111  *
1112  *
1113  *
1114  *
1115  *
1116  *
1117  *
1118  *
1119  *
1120  *
1121  *
1122  *
1123  *
1124  *
1125  *
1126  *
1127  *
1128  *
1129  *
1130  *
1131  *
1132  *
1133  *
1134  *
1135  *
1136  *
1137  *
1138  *
1139  *
1140  *
1141  *
1142  *
1143  *
1144  *
1145  *
1146  *
1147  *
1148  *
1149  *
1150  *
1151  *
1152  *
1153  *
1154  *
1155  *
1156  *
1157  *
1158  *
1159  *
1160  *
1161  *
1162  *
1163  *
1164  *
1165  *
1166  *
1167  *
1168  *
1169  *
1170  *
1171  *
1172  *
1173  *
1174  *
1175  *
1176  *
1177  *
1178  *
1179  *
1180  *
1181  *
1182  *
1183  *
1184  *
1185  *
1186  *
1187  *
1188  *
1189  *
1190  *
1191  *
1192  *
1193  *
1194  *
1195  *
1196  *
1197  *
1198  *
1199  *
1200  *
1201  *
1202  *
1203  *
1204  *
1205  *
1206  *
1207  *
1208  *
1209  *
1210  *
1211  *
1212  *
1213  *
1214  *
1215  *
1216  *
1217  *
1218  *
1219  *
1220  *
1221  *
1222  *
1223  *
1224  *
1225  *
1226  *
1227  *
1228  *
1229  *
1230  *
1231  *
1232  *
1233  *
1234  *
1235  *
1236  *
1237  *
1238  *
1239  *
1240  *
1241  *
1242  *
1243  *
1244  *
1245  *
1246  *
1247  *
1248  *
1249  *
1250  *
1251  *
1252  *
1253  *
1254  *
1255  *
1256  *
1257  *
1258  *
1259  *
1260  *
1261  *
1262  *
1263  *
1264  *
1265  *
1266  *
1267  *
1268  *
1269  *
1270  *
1271  *
1272  *
1273  *
1274  *
1275  *
1276  *
1277  *
1278  *
1279  *
1280  *
1281  *
1282  *
1283  *
1284  *
1285  *
1286  *
1287  *
1288  *
1289  *
1290  *
1291  *
1292  *
1293  *
1294  *
1295  *
1296  *
1297  *
1298  *
1299  *
1300  *
1301  *
1302  *
1303  *
1304  *
1305  *
1306  *
1307  *
1308  *
1309  *
1310  *
1311  *
1312  *
1313  *
1314  *
1315  *
1316  *
1317  *
1318  *
1319  *
1320  *
1321  *
1322  *
1323  *
1324  *
1325  *
1326  *
1327  *
1328  *
1329  *
1330  *
1331  *
1332  *
1333  *
1334  *
1335  *
1336  *
1337  *
1338  *
1339  *
1340  *
1341  *
1342  *
1343  *
1344  *
1345  *
1346  *
1347  *
1348  *
1349  *
1350  *
1351  *
1352  *
1353  *
1354  *
1355  *
1356  *
1357  *
1358  *
1359  *
1360  *
1361  *
1362  *
1363  *
1364  *
1365  *
1366  *
1367  *
1368  *
1369  *
1370  *
1371  *
1372  *
1373  *
1374  *
1375  *
1376  *
1377  *
1378  *
1379  *
1380  *
1381  *
1382  *
1383  *
1384  *
1385  *
1386  *
1387  *
1388  *
1389  *
1390  *
1391  *
1392  *
1393  *
1394  *
1395  *
1396  *
1397  *
1398  *
1399  *
1400  *
1401  *
1402  *
1403  *
1404  *
1405  *
1406  *
1407  *
1408  *
1409  *
1410  *
1411  *
1412  *
1413  *
1414  *
1415  *
1416  *
1417  *
1418  *
1419  *
1420  *
1421  *
1422  *
1423  *
1424  *
1425  *
1426  *
1427  *
1428  *
1429  *
1430  *
1431  *
1432  *
1433  *
1434  *
1435  *
1436  *
1437  *
1438  *
1439  *
1440  *
1441  *
1442  *
1443  *
1444  *
1445  *
1446  *
1447  *
1448  *
1449  *
1450  *
1451  *
1452  *
1453  *
1454  *
1455  *
1456  *
1457  *
1458  *
1459  *
1460  *
1461  *
1462  *
1463  *
1464  *
1465  *
1466  *
1467  *
1468  *
1469  *
1470  *
1471  *
1472  *
1473  *
1474  *
1475  *
1476  *
1477  *
1478  *
1479  *
1480  *
1481  *
1482  *
1483  *
1484  *
1485  *
1486  *
1487  *
1488  *
1489  *
1490  *
1491  *
1492  *
1493  *
1494  *
1495  *
1496  *
1497  *
1498  *
1499  *
1500  *
1501  *
1502  *
1503  *
1504  *
1505  *
1506  *
1507  *
1508  *
1509  *
1510  *
1511  *
1512  *
1513  *
1514  *
1515  *
1516  *
1517  *
1518  *
1519  *
1520  *
1521  *
1522  *
1523  *
1524  *
1525  *
1526  *
1527  *
1528  *
1529  *
1530  *
1531  *
1532  *
1533  *
1534  *
1535  *
1536  *
1537  *
1538  *
1539  *
1540  *
1541  *
1542  *
1543  *
1544  *
1545  *
1546  *
1547  *
1548  *
1549  *
1550  *
1551  *
1552  *
1553  *
1554  *
1555  *
1556  *
1557  *
1558  *
1559  *
1560  *
1561  *
1562  *
1563  *
1564  *
1565  *
1566  *
1567  *
1568  *
1569  *
1570  *
1571  *
1572  *
1573  *
1574  *
1575  *
1576  *
1577  *
1578  *
1579  *
1580  *
1581  *
1582  *
1583  *
1584  *
1585  *
1586  *
1587  *
1588  *
1589  *
1590  *
1591  *
1592  *
1593  *
1594  *
1595  *
1596  *
1597  *
1598  *
1599  *
1600  *
1601  *
1602  *
1603  *
1604  *
1605  *
1606  *
1607  *
1608  *
1609  *
1610  *
1611  *
1612  *
1613  *
1614  *
1615  *
1616  *
1617  *
1618  *
1619  *
1620  *
1621  *
1622  *
1623  *
1624  *
1625  *
1626  *
1627  *
1628  *
1629  *
1630  *
1631  *
1632  *
1633  *
1634  *
1635  *
1636  *
1637  *
1638  *
1639  *
1640  *
1641  *
1642  *
1643  *
1644  *
1645  *
1646  *
1647  *
1648  *
1649  *
1650  *
1651  *
1652  *
1653  *
1654  *
1655  *
1656  *
1657  *
1658  *
1659  *
1660  *
1661  *
1662  *
1663  *
1664  *
1665  *
1666  *
1667  *
1668  *
1669  *
1670  *
1671  *
1672  *
1673  *
1674  *
1675  *
1676  *
1677  *
1678  *
1679  *
1680  *
1681  *
1682  *
1683  *
1684  *
1685  *
1686  *
1687  *
1688  *
1689  *
1690  *
1691  *
1692  *
1693  *
1694  *
1695  *
1696  *
1697  *
1698  *
1699  *
1700  *
1701  *
1702  *
1703  *
1704  *
1705  *
1706  *
1707  *
1708  *
1709  *
1710  *
1711  *
1712  *
1713  *
1714  *
1715  *
1716  *
1717  *
1718  *
1719  *
1720  *
1721  *
1722  *
1723  *
1724  *
1725  *
1726  *
1727  *
1728  *
1729  *
1730  *
1731  *
1732  *
1733  *
1734  *
1735  *
1736  *
1737  *
1738  *
1739  *
1740  *
1741  *
1742  *
1743  *
1744  *
1745  *
1746  *
1747  *
1748  *
1749  *
1750  *
1751  *
1752  *
1753  *
1754  *
1755  *
1756  *
1757  *
1758  *
1759  *
1760  *
1761  *
1762  *
1763  *
1764  *
1765  *
1766  *
1767  *
1768  *
1769  *
1770  *
1771  *
1772  *
1773  *
1774  *
1775  *
1776  *
1777  *
1778  *
1779  *
1780  *
1781  *
1782  *
1783  *
1784  *
1785  *
1786  *
1787  *
1788  *
1789  *
1790  *
1791  *
1792  *
1793  *
1794  *
1795  *
1796  *
1797  *
1798  *
1799  *
1800  *
1801  *
1802  *
1803  *
1804  *
1805  *
1806  *
1807  *
1808  *
1809  *
1810  *
1811  *
1812  *
1813  *
1814  *
1815  *
1816  *
1817  *
1818  *
1819  *
1820  *
1821  *
1822  *
1823  *
1824  *
1825  *
1826  *
1827  *
1828  *
1829  *
1830  *
1831  *
1832  *
1833  *
1834  *
1835  *
1836  *
1837  *
1838  *
1839  *
1840  *
1841  *
1842  *
1843  *
1844  *
1845  *
1846  *
1847  *
1848  *
1849  *
1850  *
1851  *
1852  *
1853  *
1854  *
1855  *
1856  *
1857  *
1858  *
1859  *
1860  *
1861  *
1862  *
1863  *
1864  *
1865  *
1866  *
1867  *
1868  *
1869  *
1870  *
1871  *
1872  *
1873  *
1874  *
1875  *
1876  *
1877  *
1878  *
1879  *
1880  *
1881  *
1882  *
1883  *
1884  *
1885  *
1886  *
1887  *
1888  *
1889  *
1890  *
1891  *
1892  *
1893  *
1894  *
1895  *
1896  *
1897  *
1898  *
1899  *
1900  *
1901  *
1902  *
1903  *
1904  *
1905  *
1906  *
1907  *
1908  *
1909  *
1910  *
1911  *
1912  *
1913  *
1914  *
1915  *
1916  *
1917  *
1918  *
1919  *
1920  *
1921  *
1922  *
1923  *
1924  *
1925  *
1926  *
1927  *
1928  *
1929  *
1930  *
1931  *
1932  *
1933  *
1934  *
1935  *
1936  *
1937  *
1938  *
1939  *
1940  *
1941  *
1942  *
1943  *
1944  *
1945  *
1946  *
1947  *
1948  *
1949  *
1950  *
1951  *
1952  *
1953  *
1954  *
1955  *
1956  *
1957  *
1958  *
1959  *
1960  *
1961  *
1962  *
1963  *
1964  *
1965  *
1966  *
1967  *
1968  *
1969  *
1970  *
1971  *
1972  *
1973  *
1974  *
1975  *
1976  *
1977  *
1978  *
1979  *
1980  *
1981  *
1982  *
1983  *
1984  *
1985  *
1986  *
1987  *
1988  *
1989  *
1990  *
1991  *
1992  *
1993  *
1994  *
1995  *
1996  *
1997  *
1998  *
1999  *
2000  *
2001  *
2002  *
2003  *
2004  *
2005  *
2006  *
2007  *
2008  *
2009  *
2010  *
2011  *
2012  *
2013  *
2014  *
2015  *
2016  *
2017  *
2018  *
2019  *
2020  *
2021  *
2022  *
2023  *
2024  *
2025  *
2026  *
2027  *
2028  *
2029  *
2030  *
2031  *
2032  *
2033  *
2034  *
2035  *
2036  *
2037  *
2038  *
2039  *
2040  *
2041  *
2042  *
2043  *
2044  *
2045  *
2046  *
2047  *
2048  *
2049  *
2050  *
2051  *
2052  *
2053  *
2054  *
2055  *
2056  *
2057  *
2058  *
2059  *
2060  *
2061  *
2062  *
2063  *
2064  *
2065  *
2066  *
2067  *
2068  *
2069  *
2070  *
2071  *
2072  *
2073  *
2074  *
2075  *
2076  *
2077  *
2078  *
2079  *
2080  *
2081  *
2082  *
2083  *
2084  *
2085  *
2086  *
2087  *
2088  *
2089  *
2090  *
2091  *
2092  *
2093  *
2094  *
2095  *
2096  *
2097  *
2098  *
2099  *
2100  *
2101  *
2102  *
2103  *
2104  *
2105  *
2106  *
2107  *
2108  *
2109  *
2110  *
2111  *
2112  *
2113  *
2114  *
2115  *
2116  *
2117  *
2118  *
2119  *
2120  *
2121  *
2122  *
2123  *
2124  *
2125  *
2126  *
2127  *
2128  *
2129  *
2130  *
2131  *
2132  *
2133  *
2134  *
2135  *
2136  *
2137  *
2138  *
2139  *
2140  *
2141  *
2142  *
2143  *
2144  *
2145  *
2146  *
2147  *
2148  *
2149  *
2150  *
2151  *
2152  *
2153  *
2154  *
2155  *
2156  *
2157  *
2158  *
2159  *
2160  *
2161  *
2162  *
2163  *
2164  *
2165  *
2166  *
2167  *
2168  *
2169  *
2170  *
2171  *
2172  *
2173  *
2174  *
2175  *

```

```

396     /* Get the IPv6 addresses of host. */
397     for(i = 0; i < record_count; i++)
398     {
399         ipv6_address_ptr[i] = (NX_DNS_IPV6_ADDRESS *) (record_buffer + i *
sizeof(NX_DNS_IPV6_ADDRESS));
400
401         printf("record %d: IP address: %x:%x:%x:%x:%x:%x:%x:%x\n", i,
402             (UINT) (ipv6_address_ptr[i] -> ipv6_address[0] >> 16 & 0xFFFF),
403             (UINT) (ipv6_address_ptr[i] -> ipv6_address[0] & 0xFFFF),
404             (UINT) (ipv6_address_ptr[i] -> ipv6_address[1] >> 16 & 0xFFFF),
405             (UINT) (ipv6_address_ptr[i] -> ipv6_address[1] & 0xFFFF),
406             (UINT) (ipv6_address_ptr[i] -> ipv6_address[2] >> 16 & 0xFFFF),
407             (UINT) (ipv6_address_ptr[i] -> ipv6_address[2] & 0xFFFF),
408             (UINT) (ipv6_address_ptr[i] -> ipv6_address[3] >> 16 & 0xFFFF),
409             (UINT) (ipv6_address_ptr[i] -> ipv6_address[3] & 0xFFFF));
410     }
411
412
413
414     /*-----*/
415     /*                                     Type A
416     */
417     /* Send A type DNS Query to its DNS server and get the IPv4 address.
418     */
419     /*-----*/
420     #ifdef FEATURE_NX_IPV6
421     /* Send a DNS Client name query. Indicate the Client expects an IPv4 address
(containing an A record). If the DNS client
422     is using an IPv6 DNS server it will send this query over IPv6; otherwise
it will be sent over IPv4. */
423     status = nxd_dns_host_by_name_get(&client_dns, (UCHAR *) "www.my_example.com",
&host_ipduo_address, 400, NX_IP_VERSION_V4);
424
425     /* Check for DNS query error. */
426     if (status != NX_SUCCESS)
427     {
428         error_counter++;
429     }
430     else
431     {
432         printf("-----\n");
433         printf("Test A: \n");
434         printf("IP address: %lu.%lu.%lu.%lu\n",
435             host_ipduo_address.nxd_ip_address.v4 >> 24,
436             host_ipduo_address.nxd_ip_address.v4 >> 16 & 0xFF,
437             host_ipduo_address.nxd_ip_address.v4 >> 8 & 0xFF,
438             host_ipduo_address.nxd_ip_address.v4 & 0xFF);
439     }
440 #endif
441
442     /* Look up an IPv4 address over IPv4. */
443     status = nx_dns_host_by_name_get(&client_dns, (UCHAR *) "www.my_example.com",
&host_ip_address, 400);
444
445     /* Check for DNS query error. */
446     if (status != NX_SUCCESS)
447     {
448         error_counter++;
449     }
450     else
451     {
452         printf("-----\n");
453         printf("Test A: \n");
454         printf("IP address: %lu.%lu.%lu.%lu\n",
455             host_ip_address >> 24,
456             host_ip_address >> 16 & 0xFF,
457             host_ip_address >> 8 & 0xFF,
458             host_ip_address & 0xFF);
459     }
460
461
462     /* Look up IPv4 addresses to record multiple IPv4 addresses in record_buffer
and return the IPv4 address count. */
463     status = nx_dns_ipv4_address_by_name_get(&client_dns, (UCHAR
*) "www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
464
465

```

```

466     /* Check for DNS query error. */
467     if (status != NX_SUCCESS)
468     {
469         error_counter++;
470     }
471
472     else
473     {
474
475         printf("-----\n");
476         printf("Test A: ");
477         printf("record_count = %d \n", record_count);
478     }
479
480     /* Get the IPv4 addresses of host. */
481     for(i =0; i< record_count; i++)
482     {
483         ipv4_address_ptr[i] = (ULONG *) (record_buffer + i * sizeof(ULONG));
484         printf("record %d: IP address: %lu.%lu.%lu.%lu\n", i,
485             *ipv4_address_ptr[i] >> 24,
486             *ipv4_address_ptr[i] >> 16 & 0xFF,
487             *ipv4_address_ptr[i] >> 8 & 0xFF,
488             *ipv4_address_ptr[i] & 0xFF);
489     }
490
491
492     /*****
493     /*                                     Type A + CNAME response
494     /*
495     /* Send A type DNS Query to its DNS server and get the IPv4 address.
496     /*
497     /*****
498     /* Look up an IPv4 address over IPv4. */
499     status = nx_dns_host_by_name_get(&client_dns, (UCHAR *) "www.my_example.com",
500     &host_ip_address, 400);
501
502     /* Check for DNS query error. */
503     if (status != NX_SUCCESS)
504     {
505         error_counter++;
506     }
507
508     else
509     {
510
511         printf("-----\n");
512         printf("Test A + CNAME response: \n");
513         printf("IP address: %lu.%lu.%lu.%lu\n",
514             host_ip_address >> 24,
515             host_ip_address >> 16 & 0xFF,
516             host_ip_address >> 8 & 0xFF,
517             host_ip_address & 0xFF);
518     }
519
520     /* Look up IPv4 addresses to record multiple IPv4 addresses in record_buffer
521     and return the IPv4 address count. */
522     status = nx_dns_ipv4_address_by_name_get(&client_dns, (UCHAR
523     *) "www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
524
525     /* Check for DNS query error. */
526     if (status != NX_SUCCESS)
527     {
528         error_counter++;
529     }
530
531     else
532     {
533
534         printf("-----\n");
535         printf("Test Test A + CNAME response: ");
536         printf("record_count = %d \n", record_count);
537     }
538
539     /* Get the IPv4 addresses of host. */
540     for(i =0; i< record_count; i++)
541     {
542         ipv4_address_ptr[i] = (ULONG *) (record_buffer + i * sizeof(ULONG));
543         printf("record %d: IP address: %lu.%lu.%lu.%lu\n", i,

```

```

540             *ipv4_address_ptr[i] >> 24,
541             *ipv4_address_ptr[i] >> 16 & 0xFF,
542             *ipv4_address_ptr[i] >> 8 & 0xFF,
543             *ipv4_address_ptr[i] & 0xFF);
544     }
545
546
547
548 /*
549 /*
550 /*
551
552 #ifdef FEATURE_NX_IPV6
553
554     /* Look up a host name from an IPv6 address (reverse lookup). */
555
556     /* Create an IPv6 address for a reverse lookup. */
557     test_ipduo_server_address.nxd_ip_version = NX_IP_VERSION_V6;
558     test_ipduo_server_address.nxd_ip_address.v6[0] = 0x24046800;
559     test_ipduo_server_address.nxd_ip_address.v6[1] = 0x40050c00;
560     test_ipduo_server_address.nxd_ip_address.v6[2] = 0x00000000;
561     test_ipduo_server_address.nxd_ip_address.v6[3] = 0x00000065;
562
563     /* This will be sent over IPv6 to the DNS server who should return a PTR
564     record if it can find the information. */
565     status = nxd_dns_host_by_address_get(&client_dns, &test_ipduo_server_address,
566     &record_buffer[0], BUFFER_SIZE, 450);
567
568     /* Check for DNS query error. */
569     if (status != NX_SUCCESS)
570     {
571         error_counter++;
572     }
573
574     else
575     {
576         printf("-----\n");
577         printf("Test PTR: %s\n", record_buffer);
578     }
579 #endif
580
581 #ifdef FEATURE_NX_IPV6
582
583     /* Create an IPv4 address for the reverse lookup. If the DNS client is IPv6
584     enabled, it will send this over
585     IPv6 to the DNS server; otherwise it will send it over IPv4. In either
586     case the respective server will
587     return a PTR record if it has the information. */
588     test_ipduo_server_address.nxd_ip_version = NX_IP_VERSION_V4;
589     test_ipduo_server_address.nxd_ip_address.v4 = IP_ADDRESS(74, 125, 71, 106);
590
591     status = nxd_dns_host_by_address_get(&client_dns, &test_ipduo_server_address,
592     &record_buffer[0], BUFFER_SIZE, 450);
593
594     /* Check for DNS query error. */
595     if (status != NX_SUCCESS)
596     {
597         error_counter++;
598     }
599
600     else
601     {
602         printf("-----\n");
603         printf("Test PTR: %s\n", record_buffer);
604     }
605 #endif
606
607     /* Look up host name over IPv4. */
608     host_ip_address = IP_ADDRESS(74, 125, 71, 106);
609     status = nx_dns_host_by_address_get(&client_dns, host_ip_address,
610     &record_buffer[0], BUFFER_SIZE, 450);
611
612     /* Check for DNS query error. */
613     if (status != NX_SUCCESS)
614     {

```

```

611         error_counter++;
612     }
613
614     else
615     {
616         printf("-----\n");
617         printf("Test PTR: %s\n", record_buffer);
618     }
619
620 #ifdef NX_DNS_ENABLE_EXTENDED_RR_TYPES
621
622 /*-----*/
623 /*                                     Type CNAME
624 /*
625 /*  Send CNAME type DNS Query to its DNS server and get the canonical name .
626 /*
627 /*-----*/
628 /* Send CNAME type to record the canonical name of host in record_buffer.
629 /*
630 status = nx_dns_cname_get(&client_dns, (UCHAR *)"www.my_example.com",
631 &record_buffer[0], BUFFER_SIZE, 400);
632
633 /* Check for DNS query error. */
634 if (status != NX_SUCCESS)
635 {
636     error_counter++;
637 }
638
639 else
640 {
641     printf("-----\n");
642     printf("Test CNAME: %s\n", record_buffer);
643 }
644
645 /*-----*/
646 /*                                     Type TXT
647 /*
648 /*  Send TXT type DNS Query to its DNS server and get descriptive text.
649 /*
650 /*-----*/
651 /* Send TXT type to record the descriptive test of host in record_buffer.
652 /*
653 status = nx_dns_host_text_get(&client_dns, (UCHAR *)"www.my_example.com",
654 &record_buffer[0], BUFFER_SIZE, 400);
655
656 /* Check for DNS query error. */
657 if (status != NX_SUCCESS)
658 {
659     error_counter++;
660 }
661
662 else
663 {
664     printf("-----\n");
665     printf("Test TXT: %s\n", record_buffer);
666 }
667
668 /*-----*/
669 /*                                     Type NS
670 /*
671 /*  Send NS type DNS Query to its DNS server and get the domain name server.
672 /*
673 /*-----*/
674 /* Send NS type to record multiple name servers in record_buffer and return
675 the name server count.
676 If the DNS response includes the IPv4 addresses of name server, record it
677 similarly in record_buffer. */
678 status = nx_dns_domain_name_server_get(&client_dns, (UCHAR
679 *)"www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);

```

```

673
674     /* Check for DNS query error. */
675     if (status != NX_SUCCESS)
676     {
677         error_counter++;
678     }
679
680     else
681     {
682
683         printf("-----\n");
684         printf("Test NS: ");
685         printf("record_count = %d \n", record_count);
686     }
687
688     /* Get the name server. */
689     for(i =0; i< record_count; i++)
690     {
691         nx_dns_ns_entry_ptr[i] = (NX_DNS_NS_ENTRY *) (record_buffer + i *
692         sizeof(NX_DNS_NS_ENTRY));
693         printf("record %d: IP address: %d.%d.%d.%d\n", i,
694             nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 24,
695             nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 16 & 0xFF,
696             nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 8 & 0xFF,
697             nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address & 0xFF);
698         if(nx_dns_ns_entry_ptr[i] -> nx_dns_ns_hostname_ptr)
699             printf("hostname = %s\n", nx_dns_ns_entry_ptr[i] ->
700             nx_dns_ns_hostname_ptr);
701         else
702             printf("hostname is not set\n");
703     }
704
705     /*-----*/
706     /*                                     Type MX
707     */
708     /* Send MX type DNS Query to its DNS server and get the domain mail exchange.
709     */
710     /*-----*/
711     /* Send MX DNS query type to record multiple mail exchanges in record_buffer
712     and return the mail exchange count.
713     If the DNS response includes the IPv4 addresses of mail exchange, record
714     it similarly in record_buffer. */
715     status = nx_dns_domain_mail_exchange_get(&client_dns, (UCHAR
716     *) "www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
717
718     /* Check for DNS query error. */
719     if (status != NX_SUCCESS)
720     {
721         error_counter++;
722     }
723
724     else
725     {
726
727         printf("-----\n");
728         printf("Test MX: ");
729         printf("record_count = %d \n", record_count);
730     }
731
732     /* Get the mail exchange. */
733     for(i =0; i< record_count; i++)
734     {
735         nx_dns_mx_entry_ptr[i] = (NX_DNS_MX_ENTRY *) (record_buffer + i *
736         sizeof(NX_DNS_MX_ENTRY));
737         printf("record %d: IP address: %d.%d.%d.%d\n", i,
738             nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 24,
739             nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 16 & 0xFF,
740             nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 8 & 0xFF,
741             nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address & 0xFF);
742         printf("preference = %d \n", nx_dns_mx_entry_ptr[i] ->
743         nx_dns_mx_preference);
744         if(nx_dns_mx_entry_ptr[i] -> nx_dns_mx_hostname_ptr)
745             printf("hostname = %s\n", nx_dns_mx_entry_ptr[i] ->
746             nx_dns_mx_hostname_ptr);
747         else
748             printf("hostname is not set\n");

```

```

742     }
743
744     /*-----*/
745     /*                                     Type SRV
746     /* Send SRV type DNS Query to its DNS server and get the location of services.
747     /*-----*/
748
749     /* Send SRV DNS query type to record the location of services in
750     record_buffer and return count.
751     If the DNS response includes the IPv4 addresses of service name, record
752     it similarly in record_buffer. */
753     status = nx_dns_domain_service_get(&client_dns, (UCHAR
754     *)"www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
755
756     /* Check for DNS query error. */
757     if (status != NX_SUCCESS)
758     {
759         error_counter++;
760     }
761
762     else
763     {
764         printf("-----\n");
765         printf("Test SRV: ");
766         printf("record_count = %d \n", record_count);
767     }
768
769     /* Get the location of services. */
770     for(i =0; i< record_count; i++)
771     {
772         nx_dns_srv_entry_ptr[i] = (NX_DNS_SRV_ENTRY *) (record_buffer + i *
773         sizeof(NX_DNS_SRV_ENTRY));
774
775         printf("record %d: IP address: %d.%d.%d.%d\n", i,
776         nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 24,
777         nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 16 & 0xFF,
778         nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 8 & 0xFF,
779         nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address & 0xFF);
780         printf("port number = %d\n", nx_dns_srv_entry_ptr[i] ->
781         nx_dns_srv_port_number );
782         printf("priority = %d\n", nx_dns_srv_entry_ptr[i] ->
783         nx_dns_srv_priority );
784         printf("weight = %d\n", nx_dns_srv_entry_ptr[i] -> nx_dns_srv_weight );
785         if(nx_dns_srv_entry_ptr[i] -> nx_dns_srv_hostname_ptr)
786             printf("hostname = %s\n", nx_dns_srv_entry_ptr[i] ->
787             nx_dns_srv_hostname_ptr);
788         else
789             printf("hostname is not set\n");
790     }
791
792     /* Get the service info, NetX old API.*/
793     status = nx_dns_info_by_name_get(&client_dns, (UCHAR *)"www.my_example.com",
794     &host_address, &host_port, 200);
795
796     /* Check for DNS add server error. */
797     if (status != NX_SUCCESS)
798     {
799         error_counter++;
800     }
801
802     else
803     {
804         printf("-----\n");
805         printf("Test SRV: ");
806         printf("IP address: %d.%d.%d.%d\n",
807         host_address >> 24,
808         host_address >> 16 & 0xFF,
809         host_address >> 8 & 0xFF,
810         host_address & 0xFF);
811         printf("port number = %d\n", host_port);
812     }
813
814     /*-----*/

```

```

809 /*                                     Type SOA
*/
810 /* Send SOA type DNS Query to its DNS server and get zone of start of
authority.*/
811 /******
812
813 /* Send SOA DNS query type to record the zone of start of authority in
record_buffer. */
814 status = nx_dns_authority_zone_start_get(&client_dns, (UCHAR
*)"www.my_example.com", &record_buffer[0], BUFFER_SIZE, 400);
815
816 /* Check for DNS query error. */
817 if (status != NX_SUCCESS)
818 {
819     error_counter++;
820 }
821
822 /* Get the loc*/
823 nx_dns_soa_entry_ptr = (NX_DNS_SOA_ENTRY *) record_buffer;
824 printf("-----\n");
825 printf("Test SOA: \n");
826 printf("serial = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_serial );
827 printf("refresh = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_refresh );
828 printf("retry = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_retry );
829 printf("expire = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_expire );
830 printf("minum = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_minum );
831 if(nx_dns_soa_entry_ptr -> nx_dns_soa_host_mname_ptr)
832     printf("host mname = %s\n", nx_dns_soa_entry_ptr ->
nx_dns_soa_host_mname_ptr);
833     else
834         printf("host mame is not set\n");
835     if(nx_dns_soa_entry_ptr -> nx_dns_soa_host_rname_ptr)
836         printf("host rname = %s\n", nx_dns_soa_entry_ptr ->
nx_dns_soa_host_rname_ptr);
837     else
838         printf("host rname is not set\n");
839
840
841 #endif
842
843 /* Shutting down...*/
844
845 /* Terminate the DNS Client thread. */
846 status = nx_dns_delete(&client_dns);
847
848 return;
849 }
850
851

```



## Configuration Options

There are several configuration options for building DNS for NetX. These options can be redefined in *nxd\_dns.h*. The following list describes each in detail:

| Define                                 | Meaning   |
|--|---|
| <b>NX_DNS_TYPE_OF_SERVICE</b>          | Type of service required for the DNS UDP requests. By default, this value is defined as <code>NX_IP_NORMAL</code> for normal IP packet service.                 |
| <b>NX_DNS_TIME_TO_LIVE</b>             | Specifies the maximum number of routers a packet can pass before it is discarded. The default value is 0x80.  |
| <b>NX_DNS_FRAGMENT_OPTION</b>          | Sets the socket property to allow or disallow fragmentation of outgoing packets. The default value is <code>NX_DONT_FRAGMENT</code> .                           |
| <b>NX_DNS_QUEUE_DEPTH</b>              | Sets the maximum number of packets to store on the socket receive queue. The default value is 5.  |
| <b>NX_DNS_MAX_SERVERS</b>              | Specifies the maximum number of DNS Servers in the Client server list. The default value is 5.  |
| <b>NX_DNS_MESSAGE_MAX</b>              | The maximum DNS message size for sending DNS queries. The default value is 512, which is also the maximum size specified in RFC 1035 Section 2.3.4.             |
| <b>NX_DNS_PACKET_PAYLOAD_UNALIGNED</b> | If not defined, the size of the Client packet payload which includes the Ethernet, IP (or IPv6), and UDP headers plus the maximum DNS message size specified by |

`NX_DNS_MESSAGE_MAX`. Regardless if defined, the packet payload is the 4-byte aligned and stored in `NX_DNS_PACKET_PAYLOAD`.

#### **`NX_DNS_PACKET_POOL_SIZE`**

Size of the Client packet pool for sending DNS queries if `NX_DNS_CLIENT_USER_CREATE_PACKET_POOL` is not defined. The default value is large enough for 16 packets of payload size defined by `NX_DNS_PACKET_PAYLOAD`, and is 4-byte aligned.

#### **`NX_DNS_MAX_RETRIES`**

The maximum number of times the DNS Client will query the current DNS server before trying another server or aborting the DNS query. The default value is 3.

**`NX_DNS_MAX_RETRANS_TIMEOUT`** The maximum retransmission timeout on a DNS query to a specific DNS server. The default value is 64 seconds ( $64 * NX\_IP\_PERIODIC\_RATE$ ).

#### **`NX_DNS_IP_GATEWAY_AND_DNS_SERVER`**

If defined and the Client IPv4 gateway address is non zero, the DNS Client sets the IPv4 gateway as the Client's primary DNS server. The default value is disabled.

#### **`NX_DNS_PACKET_ALLOCATE_TIMEOUT`**

This sets the timeout option for allocating a packet from the DNS client packet pool. The default value is 1 second ( $1 * NX\_IP\_PERIODIC\_RATE$ ).

#### **`NX_DNS_CLIENT_USER_CREATE_PACKET_POOL`**

This enables the DNS Client to let the application create and set the DNS Client packet pool. By default this option is disabled, and the DNS Client creates its own packet pool in *`nx_dns_create`*.

#### **`NX_DNS_CLIENT_CLEAR_QUEUE`**

This enables the DNS Client

to clear old DNS messages off the receive queue before sending a new query. Removing these packets from previous DNS queries prevents the DNS Client socket queue from overflowing and dropping valid packets.

**NX\_DNS\_ENABLE\_EXTENDED\_RR\_TYPES**

This enables the DNS Client to query on additional DNS record types in (e.g. CNAME, NS, MX, SOA, SRV and TXT).

**NX\_DNS\_CACHE\_ENABLE**

This enables the DNS Client to store the answer records into DNS cache.

## Chapter 3

### Description of DNS Client Services

This chapter contains a description of all NetX DNS services (listed below) in alphabetic order.

In the “Return Values” section in the following API descriptions, values in **BOLD** are not affected by the **NX\_DISABLE\_ERROR\_CHECKING** define that is used to disable API error checking, while non-bold values are completely disabled.

`nx_dns_authority_zone_start_get`  
*Look up the start of a zone of authority associated with the specified host name*

`nx_dns_cache_initialize`  
*Initialize a DNS Cache.*

`nx_dns_cache_notify_clear`  
*Clear the cache full notify function.*

`nx_dns_cache_notify_set`  
*Set the cache full notify function.*

`nx_dns_cname_get`  
*Look up the canonical domain name for the input domain name alias*

`nx_dns_create`  
*Create a DNS Client instance*

`nx_dns_delete`  
*Delete a DNS Client instance*

`nx_dns_domain_name_server_get`  
*Look up the authoritative name servers for the input domain zone*

`nx_dns_domain_mail_exchange_get`  
*Look up the mail exchange associated the specified host name.*

`nx_dns_domain_service_get`

*Look up the service(s) associated with  
the specified host name*

`nx_dns_get_serverlist_size`

*Return the size of the DNS Client server list*

`nx_dns_info_by_name_get`

*Return IP address, port querying on input host name*

`nx_dns_ipv4_address_by_name_get`

*Look up the IPv4 address from the specified host name*

`nxd_dns_ipv6_address_by_name_get`

*Look up the IPv6 address from the specified host name*

`nx_dns_host_by_address_get`

*Wrapper function for `nxd_dns_host_by_address_get`  
to look up a host name from a specified IP address  
(supports only IPv4 addresses)*

`nxd_dns_host_by_address_get`

*Look up an IP address from the input host name  
(supports both IPv4 and IPv6 addresses)*

`nx_dns_host_by_name_get`

*Wrapper function for `nxd_dns_host_by_address_get`  
to look up a host name from the specified address  
(supports only IPv4 addresses)*

`nxd_dns_host_by_name_get`

*Look up an IP address from the input host name  
(supports both IPv4 and IPv6 addresses)*

`nx_dns_host_text_get`

*Look up the text data for the input domain name*

`nx_dns_packet_pool_set`

*Set the DNS Client packet pool*

`nx_dns_server_add`

*Wrapper function for `nxd_dns_server_add`  
to add a DNS Server at the specified address to the  
Client list (supports only IPv4)*

`nxd_dns_server_add`

*Add a DNS Server of the specified IP address*

*to the Client server list (supports both IPv4 or IPv6 addresses)*

`nx_dns_server_get`

*Return the DNS Server in the Client list  
(supports only IPv4 addresses)*

`nxd_dns_server_get`

*Return the DNS Server in the Client list  
(supports both IPv4 and IPv6 addresses)*

`nx_dns_server_remove`

*Wrapper function for nxd\_dns\_server\_remove  
to remove a DNS Server from the Client list*

`nxd_dns_server_remove`

*Remove a DNS Server of the specified IP address from  
the Client list (supports both IPv4 and IPv6 addresses)*

`nx_dns_server_remove_all`

*Remove all DNS Servers from the Client list*

## **nx\_dns\_authority\_zone\_start\_get**

Look up the start of the zone of authority for the input host

### **Prototype**

```
UINT nx_dns_authority_zone_start_get (NX_DNS *dns_ptr, UCHAR *host_name,
                                      VOID *record_buffer,
                                      UINT buffer_size,
                                      UINT *record_count,
                                      ULONG wait_option);
```

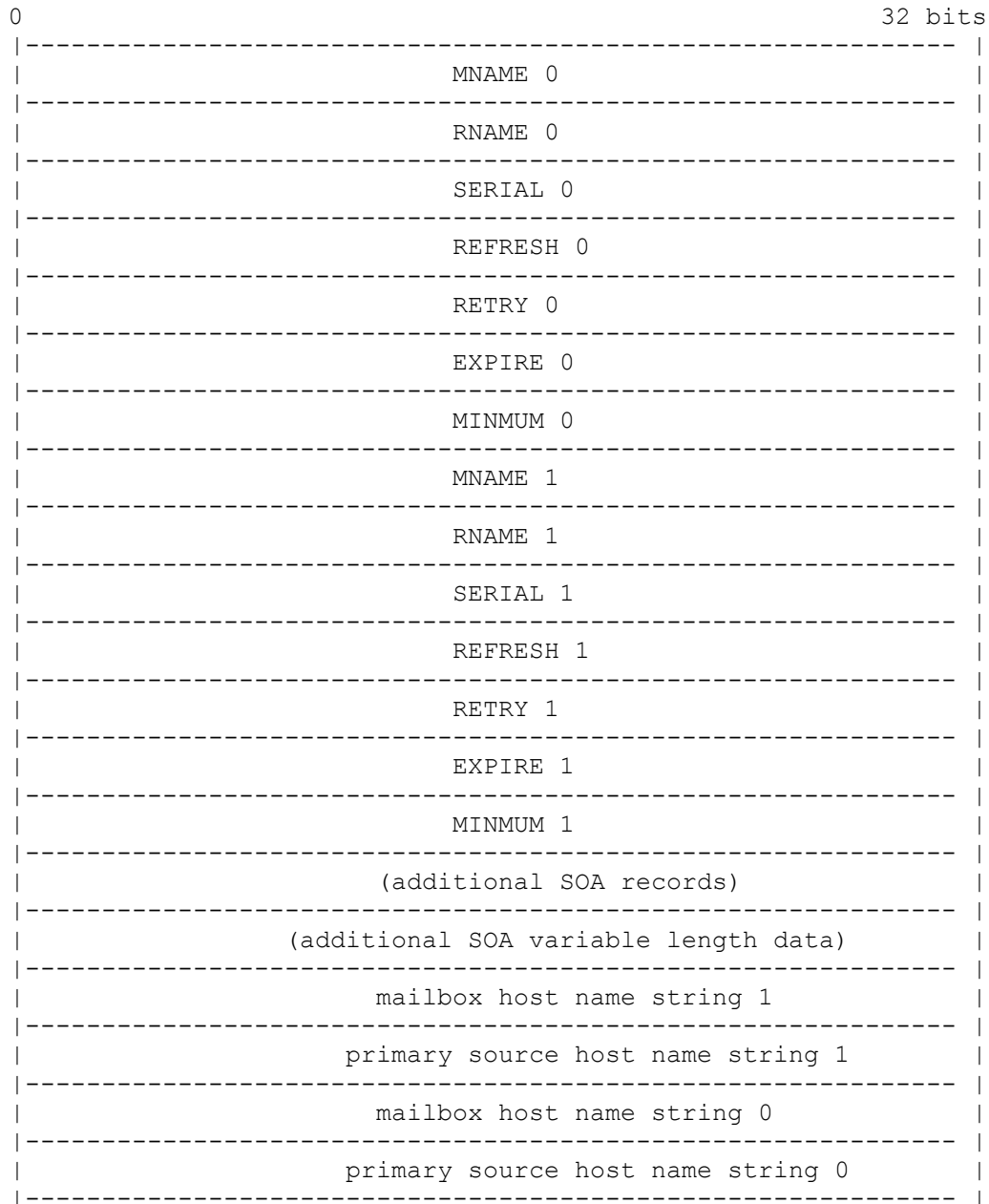
### **Description**

If `NX_DNS_ENABLE_EXTENDED_RR_TYPES` is defined, this service sends a query of type SOA with the specified domain name to obtain the start of the zone of authority for the input domain name. The DNS Client copies the SOA record(s) returned in the DNS Server response into the *record\_buffer* memory location. Note that *record\_buffer* must be 4-byte aligned to receive the data.

In NetX Duo DNS Client, the SOA record type, `NX_DNS_SOA_ENTRY`, is saved as seven 4 byte parameters, totaling 28 bytes:

|  |  |
|--|--|
| <code>nx_dns_soa_host_mname_ptr</code> | Pointer to primary source of data for this zone      |
| <code>nx_dns_soa_host_rname_ptr</code> | Pointer to mailbox responsible for this zone         |
| <code>nx_dns_soa_serial</code>         | Zone version number                                  |
| <code>nx_dns_soa_refresh</code>        | Refresh interval                                     |
| <code>nx_dns_soa_retry</code>          | Interval between SOA query retries                   |
| <code>nx_dns_soa_expire</code>         | Time duration when SOA expires                       |
| <code>nx_dns_soa_minmum</code>         | Minimum TTL field in SOA hostname DNS reply messages |

The storage of a two SOA records is shown below. The SOA records containing fixed length data are entered starting at the top of the buffer. The pointers MNAME and RNAME point to the variable length data (host names) which are stored at the bottom of the buffer. Additional SOA records are entered after the first record (“additional SOA records...”) and their variable length data is stored above the last entry’s variable length data (“additional SOA variable length data”):



If the input *record\_buffer* cannot hold all the SOA data in the server reply, the *record\_buffer* holds as many records as will fit and returns the number of records in the buffer.

With the number of SOA records returned in *\*record\_count*, the application can parse the data from *record\_buffer* and extract the start of zone authority host name strings.



## Input Parameters

|                            |  |
|----------------------------|--|
| <code>dns_ptr</code>       | Pointer to DNS Client.                         |
| <code>host_name</code>     | Pointer to host name to obtain SOA data for    |
| <code>record_buffer</code> | Pointer to location to extract SOA data into   |
| <code>buffer_size</code>   | Size of buffer to hold SOA data                |
| <code>record_count</code>  | Pointer to the number of SOA records retrieved |
| <code>wait_option</code>   | Wait option to receive DNS Server response     |

## Return Values

|   |        |                                |
|---|--------|--------------------------------|
| <b><code>NX_SUCCESS</code></b>          | (0x00) | Successfully obtained SOA data |
| <b><code>NX_DNS_NO_SERVER</code></b>    | (0xA1) | Client server list is empty    |
| <b><code>NX_DNS_QUERY_FAILED</code></b> | (0xA3) | No valid DNS response received |
| <code>NX_PTR_ERROR</code>               | (0x07) | Invalid IP or DNS pointer      |
| <code>NX_CALLER_ERROR</code>            | (0x11) | Invalid caller of this service |
| <code>NX_DNS_PARAM_ERROR</code>         | (0xA8) | Invalid non pointer input      |

## Allowed From

Threads

## Example

```

UCHAR  record_buffer[50];
UINT    record_count;
NX_DNS_SOA_ENTRY *nx_dns_soa_entry_ptr;

/* Request the start of authority zone(s) for the specified host. */
status = nx_dns_authority_zone_start_get(&client_dns, (UCHAR *)"www.my_example.com",
                                         record_buffer, sizeof(record_buffer),
                                         &record_count, 500);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}
else
{
    /* If status is NX_SUCCESS a DNS query was successfully completed and SOA data is
       returned in soa_buffer. */

    /* Set a local pointer to the SOA buffer. */
    nx_dns_soa_entry_ptr = (NX_DNS_SOA_ENTRY *) record_buffer;

    printf("-----\n");
    printf("Test SOA: \n");
    printf("serial = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_serial );
    printf("refresh = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_refresh );
    printf("retry = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_retry );
    printf("expire = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_expire );

```

```

printf("minnum = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_minnum );

if(nx_dns_soa_entry_ptr -> nx_dns_soa_host_mname_ptr)
{
    printf("host mname = %s\n",
           nx_dns_soa_entry_ptr -> nx_dns_soa_host_mname_ptr);
}
else
{
    printf("host mame is not set\n");
}

if(nx_dns_soa_entry_ptr -> nx_dns_soa_host_rname_ptr)
{
    printf("host rname = %s\n",
           nx_dns_soa_entry_ptr -> nx_dns_soa_host_rname_ptr);
}
else
{
    printf("host rname is not set\n");
}
}

```

#### [Output]

```

-----
Test SOA:
serial = 2012111212
refresh = 7200
retry = 1800
expire = 1209600
minnum = 300
host mname = ns1.www.my_example.com
host rname = dns-admin.www.my_example.com

```

## **nx\_dns\_cache\_initialize**

Initialize the DNS Cache

### **Prototype**

```
UINT nx_dns_cache_initialize(NX_DNS *dns_ptr,
                             VOID *cache_ptr, UINT cache_size);
```

### **Description**

This service creates and initializes a DNS Cache.

### **Input Parameters**

|                   |                               |
|-------------------|-------------------------------|
| <b>dns_ptr</b>    | Pointer to DNS control block. |
| <b>cache_ptr</b>  | Pointer to DNS Cache.         |
| <b>cache_size</b> | Size of DNS Cache, in bytes.  |

### **Return Values**

|                           |        |                                    |
|---------------------------|--------|------------------------------------|
| <b>NX_SUCCESS</b>         | (0x00) | DNS Cache successfully initialized |
| <b>NX_DNS_PARAM_ERROR</b> | (0xA8) | Invalid non pointer input          |
| <b>NX_DNS_CACHE_ERROR</b> | (0xB7) | Invalid Cache pointer.             |
| <b>NX_PTR_ERROR</b>       | (0x07) | Invalid DNS pointer.               |
| <b>NX_DNS_ERROR</b>       | (0xA0) | Cache is not 4-byte aligned.       |

### **Allowed From**

Threads

### **Example**

```
/* Initialize the DNS Cache. */
status = nx_dns_cache_initialize(&my_dns, &dns_cache, 2048);

/* If status is NX_SUCCESS DNS Cache was successfully initialized. */
```

## nx\_dns\_cache\_notify\_clear

## Clear the DNS Cache full notify function

## Prototype

```
UINT nx_dns_cache_notify_clear(NX_DNS *dns_ptr);
```

## Description

This service clears the cache full notify function.

## Input Parameters

|                |                               |
|----------------|-------------------------------|
| <b>dns_ptr</b> | Pointer to DNS control block. |
|----------------|-------------------------------|

## Return Values

|                    |        |                                   |
|--------------------|--------|-----------------------------------|
| <b>NX_SUCCESS</b>  | (0x00) | DNS cache notify successfully set |
| NX_DNS_PARAM_ERROR | (0xA8) | Invalid non-pointer input         |
| NX_PTR_ERROR       | (0x07) | Invalid DNS pointer.              |

## Allowed From

## Threads

## Example

```
/* Clear the DNS Cache full notify function. */
status = nx_dns_cache_notify_clear(&my_dns);

/* If status is NX_SUCCESS DNS Cache full notify function was successfully cleared. */
```

## **nx\_dns\_cache\_notify\_set**

Set the DNS Cache full notify function

### **Prototype**

```
UINT nx_dns_cache_notify_set(NX_DNS *dns_ptr,
                             VOID (*cache_full_notify_cb)(NX_DNS *dns_ptr));
```

### **Description**

This service sets the cache full notify function.

### **Input Parameters**

|                             |   |
|-----------------------------|---|
| <b>dns_ptr</b>              | Pointer to DNS control block.                               |
| <b>cache_full_notify_cb</b> | The callback function to be invoked when cache become full. |

### **Return Values**

|                           |        |                                   |
|---------------------------|--------|-----------------------------------|
| <b>NX_SUCCESS</b>         | (0x00) | DNS cache notify successfully set |
| <b>NX_DNS_PARAM_ERROR</b> | (0xA8) | Invalid non-pointer input         |
| <b>NX_PTR_ERROR</b>       | (0x07) | Invalid DNS pointer.              |

### **Allowed From**

Threads

### **Example**

```
/* Set the DNS Cache full notify function. */
status = nx_dns_cache_notify_set(&my_dns, cache_full_notify_cb);

/* If status is NX_SUCCESS DNS Cache full notify function was successfully set. */
```

## **nx\_dns\_cname\_get**

Look up the canonical name for the input hostname

### **Prototype**

```
UINT nx_dns_cname_get(NX_DNS *dns_ptr, UCHAR *host_name,
                     UCHAR *record_buffer, UINT buffer_size,
                     ULONG wait_option);
```

### **Description**

If `NX_DNS_ENABLE_EXTENDED_RR_TYPES` is defined in *nxd\_dns.h*, this service sends a query of type CNAME with the specified domain name to obtain the canonical domain name. The DNS Client copies the CNAME string returned in the DNS Server response into the *record\_buffer* memory location.

### **Input Parameters**

|                            |  |
|----------------------------|--|
| <code>dns_ptr</code>       | Pointer to DNS Client.                         |
| <code>host_name</code>     | Pointer to host name to obtain CNAME data for  |
| <code>record_buffer</code> | Pointer to location to extract CNAME data into |
| <code>buffer_size</code>   | Size of buffer to hold CNAME data              |
| <code>wait_option</code>   | Wait option to receive DNS Server response     |

### **Return Values**

|                            |        |                                  |
|----------------------------|--------|----------------------------------|
| <b>NX_SUCCESS</b>          | (0x00) | Successfully obtained CNAME data |
| <b>NX_DNS_NO_SERVER</b>    | (0xA1) | Client server list is empty      |
| <b>NX_DNS_QUERY_FAILED</b> | (0xA3) | No valid DNS response received   |
| <b>NX_PTR_ERROR</b>        | (0x07) | Invalid IP or DNS pointer        |
| <b>NX_CALLER_ERROR</b>     | (0x11) | Invalid caller of this service   |
| <b>NX_DNS_PARAM_ERROR</b>  | (0xA8) | Invalid non-pointer input        |

### **Allowed From**

Threads

### **Example**

```
CHAR          record_buffer[50];

/* Request the canonical name for the specified host. */
status = nx_dns_cname_get(&client_dns, (UCHAR *)"www.my_example.com ",
                          record_buffer, sizeof(record_buffer), 500);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}
```

```
}  
else  
{  
    /* If status is NX_SUCCESS a DNS query was successfully completed and the  
       canonical host name is returned in record_buffer. */  
  
    printf("-----\n");  
    printf("Test CNAME: %s\n", record_buffer);  
}
```

[Output]

-----  
Test CNAME: my\_example.com

## **nx\_dns\_create**

Create a DNS Client instance

### **Prototype**

```
UINT nx_dns_create(NX_DNS *dns_ptr, NX_IP *ip_ptr, CHAR *domain_name);
```

### **Description**

This service creates a DNS Client instance for the previously created IP instance.

**Important Note:** The application must ensure that the packet payload of the packet pool used by the DNS Client is large enough for the maximum 512 byte DNS message, plus UDP, IP and Ethernet headers. If the DNS Client creates its own packet pool, this is defined by NX\_DNS\_PACKET\_PAYLOAD and NX\_DNS\_PACKET\_POOL\_SIZE.

If the DNS Client application prefers to supply a previously created packet pool, the payload for IPv4 DNS Client should be 512 bytes for the maximum DNS plus 20 bytes for the IP header, 8 bytes for the UDP header and 14 bytes for the Ethernet header. For IPv6 the only difference is the IP header is 40 bytes, therefore the packet needs to accommodate the IPv6 header of 40 bytes.

### **Input Parameters**

|             |  |
|-------------|--|
| dns_ptr     | Pointer to DNS Client.                     |
| ip_ptr      | Pointer to previously created IP instance. |
| domain_name | Pointer to domain name for DNS instance.   |

### **Return Values**

|                        |        |                                |
|------------------------|--------|--------------------------------|
| <b>NX_SUCCESS</b>      | (0x00) | Successful DNS create          |
| <b>NX_DNS_ERROR</b>    | (0xA0) | DNS create error               |
| <b>NX_PTR_ERROR</b>    | (0x07) | Invalid IP or DNS pointer      |
| <b>NX_CALLER_ERROR</b> | (0x11) | Invalid caller of this service |

### **Allowed From**

Threads

### **Example**

```
/* Create a DNS Client instance. */
status = nx_dns_create(&my_dns, &my_ip, "My DNS");

/* If status is NX_SUCCESS a DNS Client instance was successfully
   created. */
```



## **nx\_dns\_delete**

Delete a DNS Client instance

### **Prototype**

```
UINT nx_dns_delete(NX_DNS *dns_ptr);
```

### **Description**

This service deletes a previously created DNS Client instance and frees up its resources. Note that if `NX_DNS_CLIENT_USER_CREATE_PACKET_POOL` is defined and the DNS Client was assigned a user defined packet pool, it is up to the application to delete the DNS Client packet pool if it no longer needs it.

### **Input Parameters**

|                      |  |
|----------------------|--|
| <code>dns_ptr</code> | Pointer to previously created DNS Client instance. |
|----------------------|--|

### **Return Values**

|                                |        |                                   |
|--------------------------------|--------|-----------------------------------|
| <b><code>NX_SUCCESS</code></b> | (0x00) | Successful DNS Client delete.     |
| <code>NX_PTR_ERROR</code>      | (0x07) | Invalid IP or DNS Client pointer. |
| <code>NX_CALLER_ERROR</code>   | (0x11) | Invalid caller of this service.   |

### **Allowed From**

Threads

### **Example**

```
/* Delete a DNS Client instance. */
status = nx_dns_delete(&my_dns);

/* If status is NX_SUCCESS the DNS Client instance was successfully
   deleted. */
```

## **nx\_dns\_domain\_name\_server\_get**

Look up the authoritative name servers for the input domain zone

### **Prototype**

```
UINT nx_dns_domain_name_server_get(NX_DNS *dns_ptr, UCHAR *host_name,
                                   VOID *record_buffer, UINT buffer_size,
                                   UINT *record_count, ULONG wait_option);
```

### **Description**

If `NX_DNS_ENABLE_EXTENDED_RR_TYPES` is defined, this service sends a query of type NS with the specified domain name to obtain the name servers for the input domain name. The DNS Client copies the NS record(s) returned in the DNS Server response into the *record\_buffer* memory location. Note that *record\_buffer* must be 4-byte aligned to receive the data.

In NetX Duo DNS Client the NS data type, `NX_DNS_NS_ENTRY`, is saved as two 4-byte parameters:

|                                     |                                       |
|-------------------------------------|---------------------------------------|
| <code>nx_dns_ns_ipv4_address</code> | Name server's IPv4 address            |
| <code>nx_dns_ns_hostname_ptr</code> | Pointer to the name server's hostname |

The buffer shown below contains four `NX_DNS_NS_ENTRY` records. The pointer to host name string in each entry points to the corresponding host name string in the bottom half of the buffer:

|          |  |                                       |  |                        |
|----------|--|---------------------------------------|--|------------------------|
| Record 0 |  | -----                                 |  |                        |
|          |  | ip_address 0                          |  | Pointer to host name 0 |
|          |  | -----                                 |  |                        |
| Record 1 |  | ip_address 1                          |  | Pointer to host name 1 |
|          |  | -----                                 |  |                        |
| Record 2 |  | ip_address 2                          |  | Pointer to host name 2 |
|          |  | -----                                 |  |                        |
| Record 3 |  | ip_address 3                          |  | Pointer to host name 3 |
|          |  | -----                                 |  |                        |
|          |  | (room for additional record entries ) |  |                        |
|          |  | -----                                 |  |                        |
|          |  | (room for additional host names )     |  |                        |
|          |  | -----                                 |  |                        |
|          |  | host name 3                           |  | host name 2            |
|          |  | -----                                 |  |                        |
|          |  | host name 1                           |  | ns_hostname 0          |
|          |  | -----                                 |  |                        |

If the input *record\_buffer* cannot hold all the NS data in the server reply, the *record\_buffer* holds as many records as will fit and returns the number of records in the buffer.

With the number of NS records returned in *\*record\_count*, the application can parse the IP address and host name of each record in the *record\_buffer*.

## Input Parameters

|                            |   |
|----------------------------|---|
| <code>dns_ptr</code>       | Pointer to DNS Client.                        |
| <code>host_name</code>     | Pointer to host name to obtain NS data for    |
| <code>record_buffer</code> | Pointer to location to extract NS data into   |
| <code>buffer_size</code>   | Size of buffer to hold NS data                |
| <code>record_count</code>  | Pointer to the number of NS records retrieved |
| <code>wait_option</code>   | Wait option to receive DNS Server response    |

## Return Values

|   |        |                                |
|---|--------|--------------------------------|
| <b><code>NX_SUCCESS</code></b>          | (0x00) | Successfully obtained NS data  |
| <b><code>NX_DNS_NO_SERVER</code></b>    | (0xA1) | Client server list is empty    |
| <b><code>NX_DNS_QUERY_FAILED</code></b> | (0xA3) | No valid DNS response received |
| <b><code>NX_DNS_PARAM_ERROR</code></b>  | (0xA8) | Invalid non-pointer input      |
| <b><code>NX_PTR_ERROR</code></b>        | (0x07) | Invalid IP or DNS pointer      |
| <b><code>NX_CALLER_ERROR</code></b>     | (0x11) | Invalid caller of this service |

## Allowed From

Threads

## Example

```
#define RECORD_COUNT    10

ULONG  record_buffer[50];
UINT   record_count;
NX_DNS_NS_ENTRY  *nx_dns_ns_entry_ptr[RECORD_COUNT];

/* Request the name server(s) for the specified host. */
status = nx_dns_domain_name_server_get(&client_dns, (UCHAR *)" www.my_example.com ",
                                       record_buffer, sizeof(record_buffer),
                                       &record_count, 500);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}

else
{
    /* If status is NX_SUCCESS a DNS query was successfully completed and NS data is
       returned in record_buffer. */

    printf("-----\n");
    printf("Test NS: ");
    printf("record_count = %d \n", record_count);
}
```

```

/* Get the name server. */
for(i =0; i< record_count; i++)
{
    nx_dns_ns_entry_ptr[i] = (NX_DNS_NS_ENTRY *)
        (record_buffer + i * sizeof(NX_DNS_NS_ENTRY));

    printf("record %d: IP address: %d.%d.%d.%d\n", i,
        nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 24,
        nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 16 & 0xFF,
        nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 8 & 0xFF,
        nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address & 0xFF);
    if(nx_dns_ns_entry_ptr[i] -> nx_dns_ns_hostname_ptr)
    {
        printf("hostname = %s\n",
            nx_dns_ns_entry_ptr[i] -> nx_dns_ns_hostname_ptr);
    }
    else
        printf("hostname is not set\n");
}
}

```

[Output]

```

-----
Test NS: record_count = 4
record 0: IP address: 192.2.2.10
hostname = ns2.www.my_example.com
record 1: IP address: 192.2.2.11
hostname = ns1.www.my_example.com
record 2: IP address: 192.2.2.12
hostname = ns3.www.my_example.com
record 3: IP address: 192.2.2.13
hostname = ns4.www.my_example.com

```

## **nx\_dns\_domain\_mail\_exchange\_get**

Look up the mail exchange(s) for the input host name

### **Prototype**

```
UINT nx_dns_domain_mail_exchange_get(NX_DNS *dns_ptr, UCHAR *host_name,
                                     VOID *record_buffer,
                                     UINT buffer_size,
                                     UINT *record_count,
                                     ULONG wait_option);
```

### **Description**

If `NX_DNS_ENABLE_EXTENDED_RR_TYPES` is defined, this service sends a query of type MX with the specified domain name to obtain the mail exchange for the input domain name. The DNS Client copies the MX record(s) returned in the DNS Server response into the *record\_buffer* memory location. Note that *record\_buffer* must be 4-byte aligned to receive the data.

In NetX Duo DNS Client, the mail exchange record type, `NX_DNS_MAIL_EXCHANGE_ENTRY`, is saved as four parameters, totaling 12 bytes:

|                                     |   |         |
|-------------------------------------|---|---------|
| <code>nx_dns_mx_ipv4_address</code> | Mail exchange IPv4 address                | 4 bytes |
| <code>nx_dns_mx_preference</code>   | Preference                                | 2 bytes |
| <code>nx_dns_mx_reserved0</code>    | Reserved                                  | 2 bytes |
| <code>nx_dns_mx_hostname_ptr</code> | Pointer to mail exchange server host name | 4 bytes |

A buffer containing four MX records is shown below. Each record contains the fixed length data from the list above. The pointer to the mail exchange server host name points to the corresponding host name at the bottom of the buffer.

|   |            |     |                      |
|---|------------|-----|----------------------|
| -----                                   |            |     |                      |
| ip address 0                            | preference | res | pointer to host name |
| -----                                   |            |     |                      |
| ip address 1                            | preference | res | pointer to host name |
| -----                                   |            |     |                      |
| ip address 2                            | preference | res | pointer to host name |
| -----                                   |            |     |                      |
| ip address 3                            | preference | res | pointer to host name |
| -----                                   |            |     |                      |
| (room for additional MX record entries) |            |     |                      |
|   |            |     |                      |
| (room for additional MX host name data) |            |     |                      |
| -----                                   |            |     |                      |
| mx_host name 3                          |            |     | mx_host name 2       |
| -----                                   |            |     |                      |
| mx_host name 1                          |            |     | mx_host name 0       |
| -----                                   |            |     |                      |

If the input *record\_buffer* cannot hold all the MX data in the server reply, the *record\_buffer* holds as many records as will fit and returns the number of records in the buffer.

With the number of MX records returned in *\*record\_count*, the application can parse the MX parameters, including the mail host name of each record in the *record\_buffer*.

## Input Parameters

|                      |   |
|----------------------|---|
| <i>dns_ptr</i>       | Pointer to DNS Client.                        |
| <i>host_name</i>     | Pointer to host name to obtain MX data for    |
| <i>record_buffer</i> | Pointer to location to extract MX data into   |
| <i>buffer_size</i>   | Size of buffer to hold MX data                |
| <i>record_count</i>  | Pointer to the number of MX records retrieved |
| <i>wait_option</i>   | Wait option to receive DNS Server response    |

## Return Values

|                            |        |                                |
|----------------------------|--------|--------------------------------|
| <b>NX_SUCCESS</b>          | (0x00) | Successfully obtained MX data  |
| <b>NX_DNS_NO_SERVER</b>    | (0xA1) | Client server list is empty    |
| <b>NX_DNS_QUERY_FAILED</b> | (0xA3) | No valid DNS response received |
| <b>NX_DNS_PARAM_ERROR</b>  | (0xA8) | Invalid non-pointer input      |
| <b>NX_PTR_ERROR</b>        | (0x07) | Invalid IP or DNS pointer      |
| <b>NX_CALLER_ERROR</b>     | (0x11) | Invalid caller of this service |

## Allowed From

Threads

## Example

```
#define MAX_RECORD_COUNT 10

ULONG record_buffer[50];
UINT record_count;
NX_DNS_MX_ENTRY *nx_dns_mx_entry_ptr[MAX_RECORD_COUNT];

/* Request the mail exchange data for the specified host. */
status = nx_dns_domain_mail_exchange_get(&client_dns, (UCHAR *) " www.my_example.com",
                                         record_buffer, sizeof(record_buffer),
                                         &record_count, 500);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}
```

```

}

else
{
    /* If status is NX_SUCCESS a DNS query was successfully completed and MX data
       is returned in record_buffer. */

    printf("-----\n");
    printf("Test MX: ");
    printf("record_count = %d \n", record_count);

    /* Get the mail exchange. */
    for(i =0; i< record_count; i++)
    {
        nx_dns_mx_entry_ptr[i] = (NX_DNS_MX_ENTRY *)
            (record_buffer + i * sizeof(NX_DNS_MX_ENTRY));

        printf("record %d: IP address: %d.%d.%d.%d\n", i,
            nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 24,
            nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 16 & 0xFF,
            nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 8 & 0xFF,
            nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address & 0xFF);

        printf("preference = %d \n ",
            nx_dns_mx_entry_ptr[i] -> nx_dns_mx_preference);

        if(nx_dns_mx_entry_ptr[i] -> nx_dns_mx_hostname_ptr)
            printf("hostname = %s\n",
                nx_dns_mx_entry_ptr[i] -> nx_dns_mx_hostname_ptr);
        else
            printf("hostname is not set\n");
    }
}

```

[Output]

```

-----
Test MX: record_count = 5
record 0: IP address: 192.2.2.10
preference = 40
hostname = alt3.aspmx.1.www.my_example.com
record 1: IP address: 192.2.2.11
preference = 50
hostname = alt4.aspmx.1.www.my_example.com
record 2: IP address: 192.2.2.12
preference = 10
hostname = aspmx.1.www.my_example.com
record 3: IP address: 192.2.2.13
preference = 20
hostname = alt1.aspmx.1.www.my_example.com
record 4: IP address: 192.2.2.14
preference = 30
hostname = alt2.aspmx.1.www.my_example.com

```





If the input *record\_buffer* cannot hold all the SRV data in the server reply, the *record\_buffer* holds as many records as will fit and returns the number of records in the buffer.

With the number of SRV records returned in *\*record\_count*, the application can parse the SRV parameters, including the server host name of each record in the *record\_buffer*.

## Input Parameters

|                            |  |
|----------------------------|--|
| <code>dns_ptr</code>       | Pointer to DNS Client.                         |
| <code>host_name</code>     | Pointer to host name to obtain SRV data for    |
| <code>record_buffer</code> | Pointer to location to extract SRV data into   |
| <code>buffer_size</code>   | Size of buffer to hold SRV data                |
| <code>record_count</code>  | Pointer to the number of SRV records retrieved |
| <code>wait_option</code>   | Wait option to receive DNS Server response     |

## Return Values

|   |        |                                |
|---|--------|--------------------------------|
| <b><code>NX_SUCCESS</code></b>          | (0x00) | Successfully obtained SRV data |
| <b><code>NX_DNS_NO_SERVER</code></b>    | (0xA1) | Client server list is empty    |
| <b><code>NX_DNS_QUERY_FAILED</code></b> | (0xA3) | No valid DNS response received |
| <b><code>NX_DNS_PARAM_ERROR</code></b>  | (0xA8) | Invalid non pointer parameter. |
| <b><code>NX_PTR_ERROR</code></b>        | (0x07) | Invalid IP or DNS pointer      |
| <b><code>NX_CALLER_ERROR</code></b>     | (0x11) | Invalid caller of this service |

## Allowed From

Threads

## Example

```
#define MAX_RECORD_COUNT 10

UCHAR record_buffer[50];
UINT record_count;
NX_DNS_SRV_ENTRY *nx_dns_srv_entry_ptr[MAX_RECORD_COUNT];

/* Request the service(s) provided by the specified host. */
status = nx_dns_domain_service_get(&client_dns, (UCHAR *)"www.my_example.com ",
                                   record_buffer, sizeof(record_buffer),
                                   &record_count, 500);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}

else
```

```

{
    /* If status is NX_SUCCESS a DNS query was successfully completed and SRV data is
       returned in record_buffer. */

    printf("-----\n");
    printf("Test SRV: ");
    printf("record_count = %d \n", record_count);

    /* Get the location of services. */
    for(i =0; i< record_count; i++)
    {
        nx_dns_srv_entry_ptr[i] = (NX_DNS_SRV_ENTRY *)
            (record_buffer + i * sizeof(NX_DNS_SRV_ENTRY));

        printf("record %d: IP address: %d.%d.%d.%d\n", i,
            nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 24,
            nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 16 & 0xFF,
            nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 8 & 0xFF,
            nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address & 0xFF);

        printf("port number = %d\n",
            nx_dns_srv_entry_ptr[i] -> nx_dns_srv_port_number );
        printf("priority = %d\n", nx_dns_srv_entry_ptr[i] -> nx_dns_srv_priority );
        printf("weight = %d\n", nx_dns_srv_entry_ptr[i] -> nx_dns_srv_weight );

        if(nx_dns_srv_entry_ptr[i] -> nx_dns_srv_hostname_ptr)
        {
            printf("hostname = %s\n",
                nx_dns_srv_entry_ptr[i] -> nx_dns_srv_hostname_ptr);
        }
        else
            printf("hostname is not set\n");
    }
}

```

#### [Output]

```

-----
Test SRV: record_count = 3
record 0: IP address: 192.2.2.10
port number = 5222
priority = 20
weight = 0
hostname = alt4.xmpp.l.www.my_example.com
record 1: IP address: 192.2.2.11
port number = 5222
priority = 5
weight = 0
hostname = xmpp.l.www.my_example.com
record 2: IP address: 192.2.2.12
port number = 5222
priority = 20
weight = 0
hostname = alt1.xmpp.l.www.my_example.com

```

## **nx\_dns\_get\_serverlist\_size**

Return the size of the DNS Client's Server list

### **Prototype**

```
UINT nx_dns_get_serverlist_size (NX_DNS *dns_ptr, UINT *size);
```

### **Description**

This service returns the number of valid DNS Servers (both IPv4 and IPv6) in the Client list.

### **Input Parameters**

|         |   |
|---------|---|
| dns_ptr | Pointer to DNS control block              |
| size    | Returns the number of servers in the list |

### **Return Values**

|                        |        |  |
|------------------------|--------|--|
| <b>NX_SUCCESS</b>      | (0x00) | DNS Server list size successfully returned |
| <b>NX_PTR_ERROR</b>    | (0x07) | Invalid IP or DNS pointer.                 |
| <b>NX_CALLER_ERROR</b> | (0x11) | Invalid caller of this service             |

### **Allowed From**

Threads

### **Example**

```
UINT my_listsize;

/* Get the number of non null DNS Servers in the Client list. */
status = nx_dns_get_serverlist_size (&my_dns, 5, &my_listsize);

/* If status is NX_SUCCESS the size of the DNS Server list was successfully
   returned. */
```

## **nx\_dns\_info\_by\_name\_get**

Return ip address and port of DNS server by host name

### **Prototype**

```
UINT nx_dns_info_by_name_get(NX_DNS *dns_ptr, UCHAR *host_name,
                             ULONG *host_address_ptr,
                             USHORT *host_port_ptr, ULONG wait_option);
```

### **Description**

This service returns the Server IP and port (service record) based on the input host name by DNS query. If a service record is not found, this routine returns a zero IP address in the input address pointer and a non-zero error status return to signal an error.

### **Input Parameters**

|                  |                                  |
|------------------|----------------------------------|
| dns_ptr          | Pointer to DNS control block     |
| host_name        | Pointer to host name buffer      |
| host_address_ptr | Pointer to address to return     |
| host_port_ptr    | Pointer to port to return        |
| wait_option      | Wait option for the DNS response |

### **Return Values**

|                            |        |   |
|----------------------------|--------|---|
| <b>NX_SUCCESS</b>          | (0x00) | DNS Server record successfully returned   |
| <b>NX_DNS_NO_SERVER</b>    | (0xA1) | No DNS Server registered with Client to send query on hostname  |
| <b>NX_DNS_QUERY_FAILED</b> | (0xA3) | DNS query failed; no response from any DNS servers in Client list or no service record is available for the input hostname. |
| <b>NX_PTR_ERROR</b>        | (0x07) | Invalid IP or DNS pointer   |
| <b>NX_CALLER_ERROR</b>     | (0x11) | Invalid caller  |

### **Allowed From**

Threads

## Example

```
ULONG ip_address
USHORT port;

/* Attempt to resolve the IP address and ports for this host name. */
status = nx_dns_info_by_name_get(&my_dns, "www.abc1234.com", &ip_address, &port,
200);

/* If status is NX_SUCCESS the DNS query was successful and the IP address and
report for the hostname are returned. */
```

## **nx\_dns\_ipv4\_address\_by\_name\_get**

Look up the IPv4 address for the input host name

### **Prototype**

```
UINT nx_dns_ipv4_address_by_name_get (NX_DNS *dns_ptr,
                                      UCHAR *host_name_ptr, VOID *buffer,
                                      UINT buffer_size,
                                      UINT *record_count,
                                      ULONG wait_option);
```

### **Description**

This service sends a query of Type A with the specified host name to obtain the IP addresses for the input host name. The DNS Client copies the IPv4 address from the A record(s) returned in the DNS Server response into the *record\_buffer* memory location. Note that *record\_buffer* must be 4-byte aligned to receive the data.

Multiple IPv4 addresses are stored in the 4-byte aligned buffer as shown below:

```
|-----|
| Address 0 | Address 1 | Address 2 | . . . . . | Address n |
|-----|
```

If the supplied buffer cannot hold all the IP address data, the remaining A records are not stored in *record\_buffer*. This enables the application to retrieve one, some or all of the available IP address data in the server reply.

With the number of A records returned in *\*record\_count* the application can parse the IPv4 address data from the *record\_buffer*.

### **Input Parameters**

|               |   |
|---------------|---|
| dns_ptr       | Pointer to DNS Client.                        |
| host_name_ptr | Pointer to host name to obtain IPv4 address   |
| buffer        | Pointer to location to extract IPv4 data into |
| buffer_size   | Size of buffer to hold IPv4 data              |
| wait_option   | Wait option to receive DNS Server response    |

### **Return Values**

|                            |        |                                 |
|----------------------------|--------|---------------------------------|
| <b>NX_SUCCESS</b>          | (0x00) | Successfully obtained IPv4 data |
| <b>NX_DNS_NO_SERVER</b>    | (0xA1) | Client server list is empty     |
| <b>NX_DNS_QUERY_FAILED</b> |        |                                 |

|                    |        |                                |
|--------------------|--------|--------------------------------|
|                    | (0xA3) | No valid DNS response received |
| NX_PTR_ERROR       | (0x07) | Invalid IP or DNS pointer      |
| NX_CALLER_ERROR    | (0x11) | Invalid caller of this service |
| NX_DNS_PARAM_ERROR | (0xA8) | Invalid non pointer parameter. |

## Allowed From

### Threads

## Example

```
#define MAX_RECORD_COUNT 20

ULONG      record_buffer[50];
UINT       record_count;
ULONG      *ipv4_address_ptr[MAX_RECORD_COUNT];

/* Request the IPv4 address for the specified host. */
status = nx_dns_ipv4_address_by_name_get(&client_dns,
                                         (UCHAR *) "www.my_example.com",
                                         record_buffer,
                                         sizeof(record_buffer), &record_count,
                                         500);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}
else
{
    /* If status is NX_SUCCESS a DNS query was successfully completed the IPv4
       address(es) is returned in record_buffer. */

    printf("-----\n");
    printf("Test A: ");
    printf("record_count = %d \n", record_count);

    /* Get the IPv4 addresses of host. */
    for(i = 0; i < record_count; i++)
    {
        ipv4_address_ptr[i] = (ULONG *) (record_buffer + i * sizeof(ULONG));
        printf("record %d: IP address: %d.%d.%d.%d\n", i,
               *ipv4_address_ptr[i] >> 24,
               *ipv4_address_ptr[i] >> 16 & 0xFF,
               *ipv4_address_ptr[i] >> 8 & 0xFF,
               *ipv4_address_ptr[i] & 0xFF);
    }
}
```

[Output]

```
-----
Test A: record_count = 5
record 0: IP address: 192.2.2.10
record 1: IP address: 192.2.2.11
record 2: IP address: 192.2.2.12
record 3: IP address: 192.2.2.13
record 4: IP address: 192.2.2.14
```

# nxd\_dns\_ipv6\_address\_by\_name\_get

Look up the IPv6 address for the input host name

## Prototype

```
UINT nxd_dns_ipv6_address_by_name_get(NX_DNS *dns_ptr,
                                      UCHAR *host_name_ptr, VOID *buffer,
                                      UINT buffer_size,
                                      UINT *record_count,
                                      ULONG wait_option);
```

## Description

This service sends a query of type AAAA with the specified domain name to obtain the IP addresses for the input domain name. The DNS Client copies the IPv6 address from the AAAA record(s) returned in the DNS Server response into the *record\_buffer* memory location. Note that *record\_buffer* must be 4-byte aligned to receive the data.

The format of IPv6 addresses stored in the 4-byte aligned buffer is shown below:

|                   |                   |                   |                   |
|-------------------|-------------------|-------------------|-------------------|
| -----             | -----             | -----             | -----             |
| IPv6_address_0[0] | IPv6_address_0[1] | IPv6_address_0[2] | IPv6_address_0[3] |
| -----             | -----             | -----             | -----             |
| IPv6_address_1[0] | IPv6_address_1[1] | IPv6_address_1[2] | IPv6_address_1[3] |
| -----             | -----             | -----             | -----             |
| IPv6_address_2[0] | IPv6_address_2[1] | IPv6_address_2[2] | IPv6_address_2[3] |
| -----             | -----             | -----             | -----             |
| IPv6_address_n[0] | IPv6_address_n[1] | IPv6_address_n[2] | IPv6_address_n[3] |
| -----             | -----             | -----             | -----             |

If the input *record\_buffer* cannot hold all the AAAA data in the server reply, the the *record\_buffer* holds as many records as will fit and returns the number of records in the buffer.

With the number of AAAA records returned in *\*record\_count*, the application can parse the IPv6 addresses from each record in the *record\_buffer*.

## Input Parameters

|               |   |
|---------------|---|
| dns_ptr       | Pointer to DNS Client.                        |
| host_name_ptr | Pointer to host name to obtain IPv6 address   |
| buffer        | Pointer to location to extract IPv6 data into |
| buffer_size   | Size of buffer to hold IPv6 data              |
| wait_option   | Wait option to receive DNS Server response    |

## Return Values

|            |        |                                 |
|------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successfully obtained IPv6 data |
|------------|--------|---------------------------------|



|                                   |                                |
|-----------------------------------|--------------------------------|
| <b>NX_DNS_NO_SERVER</b> (0xA1)    | Client server list is empty    |
| <b>NX_DNS_QUERY_FAILED</b> (0xA3) | No valid DNS response received |
| <b>NX_PTR_ERROR</b> (0x07)        | Invalid IP or DNS pointer      |
| <b>NX_CALLER_ERROR</b> (0x11)     | Invalid caller of this service |
| <b>NX_DNS_PARAM_ERROR</b> (0xA8)  | Invalid non pointer parameter. |

## Allowed From

Threads

## Example

```
#define          MAX_RECORD_COUNT  20

ULONG           record_buffer[50];
UINT            record_count;
NXD_ADDRESS     *ipv6_address_ptr[MAX_RECORD_COUNT];

/* Request the IPv4 address for the specified host. */
status = nxd_dns_ipv6_address_by_name_get(&client_dns,
                                          (UCHAR *) "www.my_example.com",
                                          record_buffer,
                                          sizeof(record_buffer),
                                          &record_count, 500);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}
else
{
    /* If status is NX_SUCCESS a DNS query was successfully completed the IPv6
       address(es) is (are) returned in record_buffer. */

    printf("-----\n");
    printf("Test AAAA: ");
    printf("record_count = %d \n", record_count);

    /* Get the IPv6 addresses of host. */
    for(i =0; i< record_count; i++)
    {
        ipv6_address_ptr[i] =
            (NX_DNS_IPV6_ADDRESS *) (record_buffer + i * sizeof(NX_DNS_IPV6_ADDRESS));

        printf("record %d: IP address: %x:%x:%x:%x:%x:%x:%x:%x\n", i,
            ipv6_address_ptr[i] -> ipv6_address[0] >>16 & 0xFFFF,
            ipv6_address_ptr[i] -> ipv6_address[0] & 0xFFFF,
            ipv6_address_ptr[i] -> ipv6_address[1] >>16 & 0xFFFF,
            ipv6_address_ptr[i] -> ipv6_address[1] & 0xFFFF,
            ipv6_address_ptr[i] -> ipv6_address[2] >>16 & 0xFFFF,
            ipv6_address_ptr[i] -> ipv6_address[2] & 0xFFFF,
            ipv6_address_ptr[i] -> ipv6_address[3] >>16 & 0xFFFF,
            ipv6_address_ptr[i] -> ipv6_address[3] & 0xFFFF);
    }
}
```

### [Output]

```
-----
Test AAAA: record_count = 1
record 0: IP address: 2001:0db8:0000:f101: 0000: 0000: 0000:01003
```

## nx\_dns\_host\_by\_address\_get

Look up a host name from an IP address

### Prototype

```
UINT nx_dns_host_by_address_get(NX_DNS *dns_ptr, ULONG ip_address,
                                ULONG *host_name_ptr,
                                ULONG max_host_name_size,
                                ULONG wait_option);
```

### Description

This service requests name resolution of the supplied IP address from one or more DNS Servers previously specified by the application. If successful, the NULL-terminated host name is returned in the string specified by *host\_name\_ptr*. This is a wrapper function for *nxd\_dns\_host\_by\_address\_get* service and does not accept IPv6 addresses.

### Input Parameters

|                    |  |
|--------------------|--|
| dns_ptr            | Pointer to previously created DNS instance.  |
| ip_address         | IP address to resolve into a name  |
| host_name_ptr      | Pointer to destination area for host name  |
| max_host_name_size | Size of destination area for host name   |
| wait_option        | Defines how long the service will wait in timer ticks for a DNS server response after each DNS query and query retry. The wait options are defined as follows: |

timeout value (0x00000001-0xFFFFFFFF)  
TX\_WAIT\_FOREVER (0xFFFFFFFF)

Selecting TX\_WAIT\_FOREVER causes the calling thread to suspend indefinitely until a DNS server responds to the request.

Selecting a numeric value (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the DNS resolution.

### Return Values

|                                    |        |  |
|------------------------------------|--------|--|
| <b>NX_SUCCESS</b>                  | (0x00) | Successful DNS resolution                        |
| <b>NX_DNS_TIMEOUT</b>              | (0xA2) | Timed out on obtaining DNS mutex                 |
| <b>NX_DNS_NO_SERVER</b>            | (0xA1) | No DNS Server address specified                  |
| <b>NX_DNS_QUERY_FAILED</b>         | (0xA3) | Received no response to query                    |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>    | (0xA4) | Null input address                               |
| <b>NX_DNS_INVALID_ADDRESS_TYPE</b> | (0xB2) | Index points to invalid address type (e.g. IPv6) |
| <b>NX_DNS_PARAM_ERROR</b>          | (0xA8) | Invalid non pointer input                        |
| <b>NX_DNS_IPV6_NOT_SUPPORTED</b>   | (0xB3) | Cannot process record with IPv6 disabled         |
| <b>NX_PTR_ERROR</b>                | (0x07) | Invalid pointer input                            |
| <b>NX_CALLER_ERROR</b>             | (0x11) | Invalid caller of this service                   |

## Allowed From

Threads

## Example

```
#define BUFFER_SIZE 200

UCHAR resolved_name[200];

/* Get the name associated with IP address 192.2.2.10. */
status = nx_dns_host_by_address_get(&my_dns, IP_ADDRESS(192.2.2.10),
                                   &resolved_name[0], BUFFER_SIZE, 450);

/* If status is NX_SUCCESS the name associated with the IP address
   can be found in the resolved_name variable. */
```

## nxd\_dns\_host\_by\_address\_get

Look up a host name from the IP address

### Prototype

```
UINT nxd_dns_host_by_address_get(NX_DNS *dns_ptr,
                                NXD_ADDRESS ip_address,
                                ULONG *host_name_ptr,
                                ULONG max_host_name_size,
                                ULONG wait_option);
```

### Description

This service requests name resolution of the IPv6 or IPv4 address in the *ip\_address* input argument from one or more DNS Servers previously specified by the application. If successful, the NULL-terminated host name is returned in the string specified by *host\_name\_ptr*.

### Input Parameters

|                    |  |
|--------------------|--|
| dns_ptr            | Pointer to previously created DNS instance.  |
| ip_address         | IP address to resolve into a name  |
| host_name_ptr      | Pointer to destination area for host name  |
| max_host_name_size | Size of destination area for host name   |
| wait_option        | Defines how long the service will wait in timer ticks for a DNS server response after each DNS query and query retry. The wait options are defined as follows: |

timeout value(0x00000001 through  
0xFFFFFFFF)  
TX\_WAIT\_FOREVER (0xFFFFFFFF)

Selecting TX\_WAIT\_FOREVER causes the calling thread to suspend indefinitely until a DNS server responds to the request.

Selecting a numeric value (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the DNS resolution.

### Return Values

|                            |        |                                  |
|----------------------------|--------|----------------------------------|
| <b>NX_SUCCESS</b>          | (0x00) | Successful DNS resolution        |
| <b>NX_DNS_TIMEOUT</b>      | (0xA2) | Timed out on obtaining DNS mutex |
| <b>NX_DNS_NO_SERVER</b>    | (0xA1) | No DNS Server address specified  |
| <b>NX_DNS_QUERY_FAILED</b> |        |                                  |

|                                  |        |  |
|----------------------------------|--------|--|
|                                  | (0xA3) | Received no response to query            |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>  | (0xA4) | Null input address                       |
| <b>NX_DNS_IPV6_NOT_SUPPORTED</b> | (0xB3) | Cannot process record with IPv6 disabled |
| <b>NX_PTR_ERROR</b>              | (0x07) | Invalid IP or DNS pointer                |
| <b>NX_CALLER_ERROR</b>           | (0x11) | Invalid caller of this service           |
| <b>NX_DNS_PARAM_ERROR</b>        | (0xA8) | Invalid non pointer input                |

## Allowed From

Threads

## Example

```

UCHAR   resolved_name[200];
NXD_ADDRESS host_address;

host_address.nxd_ip_version = NX_IP_VERSION_V6;
host_address.nxd_ip_address.v6[0] = 0x20010db8;
host_address.nxd_ip_address.v6[1] = 0x0;
host_address.nxd_ip_address.v6[2] = 0xf101;
host_address.nxd_ip_address.v6[3] = 0x108;

/* Get the name associated with the input host_address. */
status = nxd_dns_host_by_address_get(&my_dns, &host_address,
                                     resolved_name, sizeof(resolved_name), 4000);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}
else
{
    printf("-----\n");
    printf("Test PTR: %s\n", record_buffer);
}

/* If status is NX_SUCCESS the name associated with the IP address
   can be found in the resolved_name variable. */

```

## [Output]

```

-----
Test PTR: my_example.net

```

## **nx\_dns\_host\_by\_name\_get**

Look up an IP address from the host name

### **Prototype**

```
UINT nx_dns_host_by_name_get(NX_DNS *dns_ptr, ULONG *host_name,
                             ULONG *host_address_ptr, ULONG wait_option);
```

### **Description**

This service requests name resolution of the supplied name from one or more DNS Servers previously specified by the application. If successful, the associated IP address is returned in the destination pointed to by *host\_address\_ptr*. This is a wrapper function for the *nxd\_dns\_host\_by\_name\_get* service, and is limited to IPv4 address input.

### **Input Parameters**

|  |   |
|--|---|
| dns_ptr  | Pointer to previously created DNS instance.   |
| host_name_ptr  | Pointer to host name  |
| host_address_ptr   | Pointer to destination for IP address   |
| wait_option  | Defines how long the service will wait for the DNS resolution. The wait options are defined as follows: |
| timeout value  | (0x00000001 through 0xFFFFFFFF)   |
| TX_WAIT_FOREVER  | (0xFFFFFFFF)  |
| Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a DNS server responds to the request.                      |   |
| Selecting a numeric value (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the DNS resolution. |   |

### **Return Values**

|                            |        |                                 |
|----------------------------|--------|---------------------------------|
| <b>NX_SUCCESS</b>          | (0x00) | Successful DNS resolution.      |
| <b>NX_DNS_NO_SERVER</b>    | (0xA1) | No DNS Server address specified |
| <b>NX_DNS_QUERY_FAILED</b> | (0xA3) | Received no response to query   |
| <b>NX_DNS_PARAM_ERROR</b>  | (0xA8) | Invalid non pointer input       |
| <b>NX_PTR_ERROR</b>        | (0x07) | Invalid pointer input           |

**NX\_CALLER\_ERROR**      (0x11)      Invalid caller of this service

## Allowed From

Threads

## Example

```
ULONG ip_address;

/* Get the IP address for the name "www.my_example.com". */
status = nx_dns_host_by_name_get(&my_dns, "www.my_example.com", &ip_address, 4000);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}

else
{
    /* If status is NX_SUCCESS the IP address for "www.my_example.com" can be found
       in the "ip_address" variable. */

    printf("-----\n");
    printf("Test A: \n");
    printf("IP address: %d.%d.%d.%d\n",
        host_ip_address >> 24,
        host_ip_address >> 16 & 0xFF,
        host_ip_address >> 8 & 0xFF,
        host_ip_address & 0xFF);
}
```

[Output]

```
-----
Test A:
IP address: 192.2.2.10
```

## nxd\_dns\_host\_by\_name\_get

Lookup an IP address from the host name

### Prototype

```
UINT nxd_dns_host_by_name_get(NX_DNS *dns_ptr, ULONG *host_name,
                             NXD_ADDRESS *host_address_ptr,
                             ULONG wait_option, UINT lookup_type);
```

### Description

This service requests name resolution of the supplied IP address from one or more DNS Servers previously specified by the application. If successful, the associated IP address is returned in an NXD\_ADDRESS pointed to by host\_address\_ptr. If the caller specifically sets the lookup\_type input to NX\_IP\_VERSION\_V6, this service will send out query for a host IPv6 address (AAAA record). If the caller specifically sets the lookup\_type input to NX\_IP\_VERSION\_V4, this service will send out query for a host IPv4 address (A record).

### Input Parameters

|                  |  |
|------------------|--|
| dns_ptr          | Pointer to previously created DNS Client instance.   |
| host_name_ptr    | Pointer to host name to find an IP address of  |
| host_address_ptr | Pointer to destination for NXD_ADDRESS containing the IP address   |
| lookup_type      | Indicate type of lookup (A vs AAAA).   |
| wait_option      | Defines how long the service will wait in timer ticks for the DNS Server response for each query transmission and retransmission. The wait options are defined as follows: |

|                 |                                 |
|-----------------|---------------------------------|
| timeout value   | (0x00000001 through 0xFFFFFFFF) |
| TX_WAIT_FOREVER | (0xFFFFFFFF)                    |

Selecting TX\_WAIT\_FOREVER causes the calling thread to suspend indefinitely until a DNS Server responds to the request.

Selecting a numeric value (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the DNS resolution.

### Return Values



|                                  |        |  |
|----------------------------------|--------|--|
| <b>NX_SUCCESS</b>                | (0x00) | Successful DNS resolution.               |
| <b>NX_DNS_NO_SERVER</b>          | (0xA1) | No DNS Server address specified          |
| <b>NX_DNS_QUERY_FAILED</b>       | (0xA3) | Received no response to query            |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>  | (0xA4) | Null input address                       |
| <b>NX_DNS_IPV6_NOT_SUPPORTED</b> | (0xB3) | Cannot process record with IPv6 disabled |
| <b>NX_PTR_ERROR</b>              | (0x07) | Invalid pointer input                    |
| <b>NX_CALLER_ERROR</b>           | (0x11) | Invalid caller of this service           |
| <b>NX_DNS_PARAM_ERROR</b>        | (0xA8) | Invalid non pointer input                |

## Allowed From

Threads

## Example

```

NXD_ADDRESS host_ipduo_address;

/* Create an AAAA query to obtain the IPv6 address for the host "www.my_example.com".
*/
status = nxd_dns_host_by_name_get(&my_dns, "www.my_example.com", &
host_ipduo_address, 4000,
                                NX_IP_VERSION_V6);

if (status != NX_SUCCESS)
{
    error_counter++;
}
else
{
    /* If status is NX_SUCCESS the IP address for "www.my_example.com" can be found in
    the "ip_address" variable. */

    printf("-----\n");
    printf("Test AAAA: \n");

    printf("IP address: %x:%x:%x:%x:%x:%x:%x:%x\n",
        host_ipduo_address.nxd_ip_address.v6[0] >>16 & 0xFFFF,
        host_ipduo_address.nxd_ip_address.v6[0] & 0xFFFF,
        host_ipduo_address.nxd_ip_address.v6[1] >>16 & 0xFFFF,
        host_ipduo_address.nxd_ip_address.v6[1] & 0xFFFF,
        host_ipduo_address.nxd_ip_address.v6[2] >>16 & 0xFFFF,
        host_ipduo_address.nxd_ip_address.v6[2] & 0xFFFF,
        host_ipduo_address.nxd_ip_address.v6[3] >>16 & 0xFFFF,
        host_ipduo_address.nxd_ip_address.v6[3] & 0xFFFF);
}

```

[Output]

```

-----
Test AAAA:
IP address: 2607:f8b0:4007:800:0:0:0:1008

```

Another example of using this time service, this time using IPv4 addresses and A record types, is shown below:

```

/* Create a query to obtain the IPv4 address for the host "www.my_example.com". */
status = nxd_dns_host_by_name_get(&my_dns, "www.my_example.com", &ip_address, 4000,
                                  NX_IP_VERSION_V4);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}
else
{
    /* If status is NX_SUCCESS the IP address for "www.my_example.com" can be found
       in the "ip_address" variable. */

    printf("-----\n");
    printf("Test A: \n");
    printf("IP address: %d.%d.%d.%d\n",
           host_ipduo_address.nxd_ip_address.v4 >> 24,
           host_ipduo_address.nxd_ip_address.v4 >> 16 & 0xFF,
           host_ipduo_address.nxd_ip_address.v4 >> 8 & 0xFF,
           host_ipduo_address.nxd_ip_address.v4 & 0xFF);
}

```

[Output]

```

-----
Test A:
IP address: 192.2.2.10

```

## **nx\_dns\_host\_text\_get**

Look up the text string for the input domain name

### **Prototype**

```
UINT nx_dns_host_text_get(NX_DNS *dns_ptr, UCHAR *host_name,
                          UCHAR *record_buffer,
                          UINT buffer_size, ULONG wait_option);
```

### **Description**

This service sends a query of type TXT with the specified domain name and buffer to obtain the arbitrary string data.

The DNS Client copies the text string in the TXT record in the DNS Server response into the *record\_buffer* memory location. Note that *record\_buffer* does not need to be 4-byte aligned to receive the data.

### **Input Parameters**

|               |  |
|---------------|--|
| dns_ptr       | Pointer to DNS Client.                       |
| host_name     | Pointer to name of host to search on         |
| record_buffer | Pointer to location to extract TXT data into |
| buffer_size   | Size of buffer to hold TXT data              |
| wait_option   | Wait option to receive DNS Server response   |

### **Return Values**

|                            |        |                                  |
|----------------------------|--------|----------------------------------|
| <b>NX_SUCCESS</b>          | (0x00) | Successfully TXT string obtained |
| <b>NX_DNS_NO_SERVER</b>    | (0xA1) | Client server list is empty      |
| <b>NX_DNS_QUERY_FAILED</b> | (0xA3) | No valid DNS response received   |
| <b>NX_PTR_ERROR</b>        | (0x07) | Invalid pointer input            |
| <b>NX_CALLER_ERROR</b>     | (0x11) | Invalid caller of this service   |
| <b>NX_DNS_PARAM_ERROR</b>  | (0xA8) | Invalid non pointer input        |

### **Allowed From**

Threads

## Example

```

CHAR          record_buffer[50];

/* Request the text string for the specified host. */
status = nx_dns_host_text_get(&client_dns, (UCHAR *)"www.my_example.com",
                              record_buffer,
                              sizeof(record_buffer), 500);

/* Check for DNS query error. */
if (status != NX_SUCCESS)
{
    error_counter++;
}
else
{
    /* If status is NX_SUCCESS a DNS query was successfully completed and the text
       string is returned in record_buffer. */

    printf("-----\n");
    printf("Test TXT:\n %s\n", record_buffer);
}

```

[Output]

```

-----
Test TXT:
v=spf1 include:_www.my_example.com ip4:192.2.2.10/31 ip4:192.2.2.11/31 ~all

```

## **nx\_dns\_packet\_pool\_set**

Set the DNS Client packet pool

### **Prototype**

```
UINT nx_dns_packet_pool_set(NX_DNS *dns_ptr, NX_PACKET_POOL *pool_ptr);
```

### **Description**

This service sets a previously created packet pool as the DNS Client packet pool. The DNS Client will use this packet pool to send DNS queries, so the packet payload should not be less than `NX_DNS_PACKET_PAYLOAD` which includes the Ethernet, IP and UDP headers and is defined in *nxd\_dns.h*. Note that when the DNS Client is deleted, the packet pool is not deleted with it and it is the responsibility of the application to delete the packet pool when it no longer needs it.

Note: this service is only available if the configuration option `NX_DNS_CLIENT_USER_CREATE_PACKET_POOL` is defined in *nxd\_dns.h*

### **Input Parameters**

|                       |  |
|-----------------------|--|
| <code>dns_ptr</code>  | Pointer to previously created DNS Client instance. |
| <code>pool_ptr</code> | Pointer to previously created packet pool          |

### **Return Values**

|                                     |        |                                       |
|-------------------------------------|--------|---------------------------------------|
| <b><code>NX_SUCCESS</code></b>      | (0x00) | Successful completion.                |
| <b><code>NX_NOT_ENABLED</code></b>  | (0x14) | Client not configured for this option |
| <b><code>NX_PTR_ERROR</code></b>    | (0x07) | Invalid IP or DNS Client pointer.     |
| <b><code>NX_CALLER_ERROR</code></b> | (0x11) | Invalid caller of this service.       |

### **Allowed From**

Threads

## Example

```
NXD_DNS my_dns;
NX_PACKET_POOL client_pool;
NX_IP *ip_ptr;

/* Create the DNS Client. */
status = nx_dns_create(&my_dns, ip_ptr, "My DNS Client");

/* Create a packet pool for the DNS Client. */
status = nx_packet_pool_create(&client_pool, "DNS Client Packet Pool",
                               NX_DNS_PACKET_PAYLOAD, free_mem_pointer,
                               NX_DNS_PACKET_POOL_SIZE);

/* Set the DNS Client packet pool. */
status = nx_dns_packet_pool_set(&my_dns, &client_pool);

/* If status is NX_SUCCESS the DNS Client packet pool was successfully set. */
```

## **nx\_dns\_server\_add**

Add DNS Server IP Address

### **Prototype**

```
UINT nx_dns_server_add(NX_DNS *dns_ptr, ULONG server_address);
```

### **Description**

This service adds an IPv4 DNS Server to the server list.

### **Input Parameters**

|                |                               |
|----------------|-------------------------------|
| dns_ptr        | Pointer to DNS control block. |
| server_address | IP address of DNS Server      |

### **Return Values**

|                                  |        |   |
|----------------------------------|--------|---|
| <b>NX_SUCCESS</b>                | (0x00) | Server successfully added                     |
| <b>NX_DNS_DUPLICATE_ENTRY</b>    |        |   |
| <b>NX_NO_MORE_ENTRIES</b>        | (0x17) | No more DNS Servers<br>Allowed (list is full) |
| <b>NX_DNS_PARAM_ERROR</b>        | (0xA8) | Invalid non pointer input                     |
| <b>NX_DNS_IPV6_NOT_SUPPORTED</b> | (0xB3) | Cannot process record with<br>IPv6 disabled   |
| <b>NX_PTR_ERROR</b>              | (0x07) | Invalid pointer input                         |
| <b>NX_CALLER_ERROR</b>           | (0x11) | Invalid caller of this service                |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>  | (0xA4) | Null server address input                     |

### **Allowed From**

Threads

### **Example**

```
/* Add a DNS Server at IP address 202.2.2.13. */
status = nx_dns_server_add(&my_dns, IP_ADDRESS(202,2,2,13));

/* If status is NX_SUCCESS a DNS Server was successfully added. */
```

## **nxd\_dns\_server\_add**

Add DNS Server to the Client list

### **Prototype**

```
UINT nxd_dns_server_add(NX_DNS *dns_ptr, NXD_ADDRESS *server_address);
```

### **Description**

This service adds the IP address of a DNS server to the DNS Client server list. The `server_address` may be either an IPv4 or IPv6 address. If the Client wishes to be able to access the same server by either its IPv4 address or IPv6 address it should add both IP addresses as entries to the server list.

### **Input Parameters**

|                       |  |
|-----------------------|--|
| <b>dns_ptr</b>        | Pointer to DNS control block.  |
| <b>server_address</b> | Pointer to the NXD_ADDRESS containing the server IP address of DNS Server. |

### **Return Values**

|                                    |        |  |
|------------------------------------|--------|--|
| <b>NX_SUCCESS</b>                  | (0x00) | Server successfully added                        |
| <b>NX_DNS_DUPLICATE_ENTRY</b>      |        |  |
| <b>NX_NO_MORE_ENTRIES</b>          | (0x17) | No more DNS Servers allowed (list is full)       |
| <b>NX_DNS_IPV6_NOT_SUPPORTED</b>   | (0xB3) | Cannot process record with IPv6 disabled         |
| <b>NX_DNS_PARAM_ERROR</b>          | (0xA8) | Invalid non pointer input                        |
| <b>NX_PTR_ERROR</b>                | (0x07) | Invalid pointer input                            |
| <b>NX_CALLER_ERROR</b>             | (0x11) | Invalid caller of this service                   |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>    | (0xA4) | Null server address input                        |
| <b>NX_DNS_INVALID_ADDRESS_TYPE</b> | (0xB2) | Index points to invalid address type (e.g. IPv6) |

### **Allowed From**

Threads



## Example

```
NXD_ADDRESS server_address;

server_address.nxd_ip_version = NX_IP_VERSION_V6;
server_address.nxd_ip_address.v6[0] = 0x20010db8;
server_address.nxd_ip_address.v6[1] = 0x0;
server_address.nxd_ip_address.v6[2] = 0xf101;
server_address.nxd_ip_address.v6[3] = 0x108;

/* Add a DNS Server with the IP address pointed to by the server_address input. */
status = nxd_dns_server_add(&my_dns, &server_address);

/* If status is NX_SUCCESS a DNS Server was successfully added. */
```

## **nx\_dns\_server\_get**

Return an IPv4 DNS Server from the Client list

### **Prototype**

```
UINT nx_dns_server_get(NX_DNS *dns_ptr, UINT index,
                      ULONG *dns_server_address);
```

### **Description**

This service returns the IPv4 DNS Server address from the server list at the specified index. Note that the index is zero based. If the input index exceeds the size of the DNS Client list, an IPv6 address is found at that index or a null address is found at the specified index, an error is returned. The *nx\_dns\_get\_serverlist\_size* service may be called first obtain the number of DNS servers in the Client list.

This service does only supports IPv4 addresses. It calls the *nxd\_dns\_server\_get* service which supports both IPv4 and IPv6 addresses.

### **Input Parameters**

|                    |   |
|--------------------|---|
| dns_ptr            | Pointer to DNS control block            |
| index              | Index into DNS Client's list of servers |
| dns_server_address | Pointer to IP address of DNS Server     |

### **Return Values**

|                                    |        |  |
|------------------------------------|--------|--|
| <b>NX_SUCCESS</b>                  | (0x00) | Successful server returned                       |
| <b>NX_DNS_SERVER_NOT_FOUND</b>     | (0xA9) | Index points to empty slot                       |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>    | (0xA4) | Index points to Null address                     |
| <b>NX_DNS_INVALID_ADDRESS_TYPE</b> | (0xB2) | Index points to invalid address type (e.g. IPv6) |
| <b>NX_DNS_PARAM_ERROR</b>          | (0xA8) | Invalid non-pointer input                        |
| <b>NX_PTR_ERROR</b>                | (0x07) | Invalid IP or DNS pointer.                       |
| <b>NX_CALLER_ERROR</b>             | (0x11) | Invalid caller of this service                   |

### **Allowed From**

Threads

## Example

```
ULONG my_server_address;

/* Get the DNS Server at index 5 (zero based) into the Client list. */
status = nx_dns_server_get(&my_dns, 5, &my_server_address);

/* If status is NX_SUCCESS a DNS Server was successfully
   returned. */
```

## **nxd\_dns\_server\_get**

Return a DNS Server from the Client list

### **Prototype**

```
UINT nxd_dns_server_get(NX_DNS *dns_ptr, UINT index,
                       NXD_ADDRESS *dns_server_address);
```

### **Description**

This service returns the DNS Server IP address from the server list at the specified index. Note that the index is zero based. If the input index exceeds the size of the DNS Client list, or a null address is found at the specified index, an error is returned. The *nxd\_dns\_get\_serverlist\_size* service may be called first to obtain the number of DNS servers in the server list.

This service supports IPv4 and IPv6 addresses.

### **Input Parameters**

|                    |   |
|--------------------|---|
| dns_ptr            | Pointer to DNS control block            |
| index              | Index into DNS Client's list of servers |
| dns_server_address | Pointer to IP address of DNS Server     |

### **Return Values**

|                                    |        |  |
|------------------------------------|--------|--|
| <b>NX_SUCCESS</b>                  | (0x00) | Successfully returned server IP address          |
| <b>NX_DNS_SERVER_NOT_FOUND</b>     | (0xA9) | Index points to empty slot                       |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>    | (0xA4) | Index points to null server address              |
| <b>NX_DNS_INVALID_ADDRESS_TYPE</b> | (0xB2) | Index points to invalid address type (e.g. IPv6) |
| <b>NX_DNS_PARAM_ERROR</b>          | (0xA8) | Invalid non pointer input                        |
| <b>NX_PTR_ERROR</b>                | (0x07) | Invalid IP or DNS pointer.                       |
| <b>NX_CALLER_ERROR</b>             | (0x11) | Invalid caller of this service                   |

### **Allowed From**

Threads

## Example

```
NXD_ADDRESS my_server_address;

/* Get the DNS Server at index 5 (zero based) into the Client list. */
status = nxd_dns_server_get(&my_dns, 5, &my_server_address);

/* If status is NX_SUCCESS a DNS Server was successfully
   returned. */
```

## **nx\_dns\_server\_remove**

Remove an IPv4 DNS Server from the Client list

### **Prototype**

```
UINT nx_dns_server_remove(NX_DNS *dns_ptr, ULONG server_address);
```

### **Description**

This service removes an IPv4 DNS Server from the Client list. It is a wrapper function for *nxd\_dns\_server\_remove*.

### **Input Parameters**

|                       |                               |
|-----------------------|-------------------------------|
| <b>dns_ptr</b>        | Pointer to DNS control block. |
| <b>server_address</b> | IP address of DNS Server.     |

### **Return Values**

|                                  |        |  |
|----------------------------------|--------|--|
| <b>NX_SUCCESS</b>                | (0x00) | DNS Server successfully removed          |
| <b>NX_DNS_SERVER_NOT_FOUND</b>   | (0xA9) | Server not in Client list                |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>  | (0xA4) | Null server address input                |
| <b>NX_DNS_IPV6_NOT_SUPPORTED</b> | (0xB3) | Cannot process record with IPv6 disabled |
| <b>NX_PTR_ERROR</b>              | (0x07) | Invalid IP or DNS pointer.               |
| <b>NX_CALLER_ERROR</b>           | (0x11) | Invalid caller of this service           |

### **Allowed From**

Threads

## Example

```
/* Remove the DNS Server at IP address is 202.2.2.13. */
status = nx_dns_server_remove(&my_dns, IP_ADDRESS(202,2,2,13));

/* If status is NX_SUCCESS a DNS Server was successfully
   removed. */
```

## **nxd\_dns\_server\_remove**

Remove a DNS Server from the Client list

### **Prototype**

```
UINT nxd_dns_server_remove(NX_DNS *dns_ptr, NXD_ADDRESS *server_address);
```

### **Description**

This service removes a DNS Server of the specified IP address from the Client list. The input IP address accepts both IPv4 and IPv6 addresses. After the server is removed, the remaining servers move down one index in the list to fill the vacated slot.

### **Input Parameters**

|                       |  |
|-----------------------|--|
| <b>dns_ptr</b>        | Pointer to DNS control block.  |
| <b>server_address</b> | Pointer to DNS Server NXD_ADDRESS data containing server IP address. |

### **Return Values**

|                                    |        |  |
|------------------------------------|--------|--|
| <b>NX_SUCCESS</b>                  | (0x00) | DNS Server successfully removed                  |
| <b>NX_DNS_SERVER_NOT_FOUND</b>     | (0xA9) | Server not in Client list                        |
| <b>NX_DNS_BAD_ADDRESS_ERROR</b>    | (0xA4) | Null server address input                        |
| <b>NX_DNS_IPV6_NOT_SUPPORTED</b>   | (0xB3) | Cannot process record with IPv6 disabled         |
| <b>NX_PTR_ERROR</b>                | (0x07) | Invalid IP or DNS pointer.                       |
| <b>NX_CALLER_ERROR</b>             | (0x11) | Invalid caller of this service                   |
| <b>NX_DNS_INVALID_ADDRESS_TYPE</b> | (0xB2) | Index points to invalid address type (e.g. IPv6) |

### **Allowed From**

Threads

### **Example**



```
NXD_ADDRESS server_address;

server_address.nxd_ip_version = NX_IP_VERSION_V6;
server_address.nxd_ip_address.v6[0] = 0x20010db8;
server_address.nxd_ip_address.v6[1] = 0x0;
server_address.nxd_ip_address.v6[2] = 0xf101;
server_address.nxd_ip_address.v6[3] = 0x108;

/* Remove the DNS Server at the specified IP address from the Client list. */
status = nxd_dns_server_remove(&my_dns,&server_ADDRESS);

/* If status is NX_SUCCESS a DNS Server was successfully removed. */
```

## **nx\_dns\_server\_remove\_all**

Remove all DNS Servers from the Client list

### **Prototype**

```
UINT nx_dns_server_remove_all(NX_DNS *dns_ptr);
```

### **Description**

This service removes all DNS Servers from the Client list.

### **Input Parameters**

|                |                               |
|----------------|-------------------------------|
| <b>dns_ptr</b> | Pointer to DNS control block. |
|----------------|-------------------------------|

### **Return Values**

|                        |        |                                   |
|------------------------|--------|-----------------------------------|
| <b>NX_SUCCESS</b>      | (0x00) | DNS Servers successfully removed  |
| <b>NX_DNS_ERROR</b>    | (0xA0) | Unable to obtain protection mutex |
| <b>NX_PTR_ERROR</b>    | (0x07) | Invalid IP or DNS pointer.        |
| <b>NX_CALLER_ERROR</b> | (0x11) | Invalid caller of this service    |

### **Allowed From**

Threads

### **Example**

```
/* Remove all DNS Servers from the Client list. */
status = nx_dns_server_remove_all(&my_dns);

/* If status is NX_SUCCESS all DNS Servers were successfully removed. */
```