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

Part Number: 000-1051
Revision 5.2

Contents

Chapter 1 Introduction to the NetX Duo DNS Client 4
DNS Client Setup 4
DNS Messages 4
NetX Duo DNS Client Requirements 5
NetX Duo DNS Client Limitations 5
DNS RFCs 6
Chapter 2 Installation and Use of NetX Duo DNS Client 7
Product Distribution 7
DNS Client Installation 7
Using the DNS Client 7
Small Example System for NetX Duo DNS Client 8
Configuration Options 15
Chapter 3 Description of DNS Client Services 17
nx_dns_create 19
nx_dns_packet_pool_set 20
nx_dns_delete 22
nx_dns_host_by_address_get 23
nxd_dns_host_by_address_get 25
nx_dns_host_by_name_get 27
nxd_dns_host_by_name_get 29
nx_dns_server_add 31
nxd_dns_server_add 32
nx_dns_server_remove 33
nxd_dns_server_remove 34

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 contain 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 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 IP instance has already been created. In addition, a gateway IP address or at least one DNS Server IP addresses must be known. If this 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.

Upon creation, the DNS Client inherits the gateway IP address from the IP structure. By default, it is assumed that the network gateway is also a DNS Server, however this is a configurable option (see **Configuration Options** in Chapter 2 for more details). Additional DNS Servers may be specified through the ***nxd_dns_server_add*** service. The NetX Duo DNS Client supports the ***nx_dns_server_add*** for adding IPv4 DNS Servers to the DNS Client but encourages developers to use ***nxd_dns_server_add*** which accepts both IPv4 and IPv6 DNS server addresses.

DNS Messages

The DNS has a very simple mechanism for obtaining mapping between logical 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 DNS Client server list. If the DNS Server has such a mapping, it replies to the DNS Client using a DNS response message that contains the requested

mapping information. If the DNS Server does not respond, the next DNS Server is sent the same DNS message. This process continues until a successful response is received or until all known DNS servers have been queried.

NetX Duo DNS Client can perform both IPv6 address lookups (type AAAA) and IPv4 address lookups (type A) by specifying which version if IP in the ***nxd_dns_host_by_name_get*** call. The DNS Client can perform reverse lookups of IP addresses (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*** services which are limited to IPv4 network communication, but developers are encouraged to port existing DNS Client applications to the 'duo' services.

NetX Duo DNS Client Requirements

DNS utilizes the UDP protocol on port number 53 to send requests and field responses. Therefore UDP must be enabled in NetX Duo. The host application must create an IP instance and enable ICMPv6 and IPv6 in NetX Duo. The DNS Client creates its own packet pool in the ***nx_dns_create*** call. The size of the packet pool is determined by the configurable options NX_DNS_PACKET_POOL_SIZE and NX_DNS_MESSAGE_MAX. Packets for receiving DNS packets are allocated from either the IP default packet pool or driver receive packet pool depending on application design.

If the configurable option NX_DNS_CLIENT_USER_CREATE_PACKET_POOL is enabled, the host application can set the DNS Client packet pool with an already created packet pool using the ***nx_dns_packet_pool_set*** service. Note that the packet payload size must be at least NX_DNS_PACKET_PAYLOAD.

After creating the DNS Client, the host application must add one or more servers in its DNS server list. To add DNS servers, the host application use the previously mentioned ***nxd_dns_server_add*** service. If the NX_DNS_IP_GATEWAY_SERVER option is enabled, the IP instance gateway is automatically added as the primary DNS server. See the example application for details how to do this.

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

NetX Duo DNS Client Limitations

The DNS Client allows support for one DNS request at a time. Threads attempting to make another DNS request are temporarily blocked until the original DNS request is complete.

The NetX Duo DNS Client does not process CNAME records or use data from authoritative answers to forward additional DNS queries to other DNS Servers. The exception is if an application creates a DNS request containing expecting an IPv6 address, and the Server response indicates the host ip address is an IPv4 address. The DNS Client will automatically resend query to the DNS Server for an IPv4 host address.

DNS RFCs

NetX DNS is compliant with RFC1034, RFC1035, RFC1480, and related RFCs.

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:

nxd_dns.h	Header file for NetX Duo DNS Client
nxd_dns.c	C Source file for NetX Duo DNS Client
nxd_dns.pdf	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)	**NetX Duo DNS API service (IPv4 and IPv6 supported)**
nx_dns_get_host_by_name	*nxd_dns_get_host_by_name*
nx_dns_get_host_by_address	*nxd_dns_get_host_by_address*
nx_dns_add_server	*nxd_dns_add_server*
nx_dns_ remove_server	*nxd_dns_ remove_server*

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

An example of how easy it is to use NetX Duo DNS Client is described in Figure 1.1 that appears below. In this example, the demo program includes file *nxd_dns.h* in at line 5. There is a conditional define for determining which NetX Duo environment is being used with the DNS Client in lines 27-30. This is necessary for minor differences in NetX Duo versions for setting the host link local and global addresses. A thread for making NetX Duo DNS Client API calls is created in line 66. Next, a packet pool for the Client IP instance for internal NetX Duo operations is created in line 76, followed by creating the Client IP instance in line 89. UDP and ARP (which is required by NetX, not NetX Duo) is enabled in lines 103 and 115 respectively.

The Client thread entry function is then allowed to run. It initially relinquishes control to the system to allow the IP task thread be initialize by the network driver. It then enables IPv6 and ICMPv6 in NetX Duo (lines 153 and 162). The host application registers its link local and global IPv6 addresses with NetX Duo in lines 180-212. At this point, it waits while NetX Duo validates these IP addresses on line 215. See Chapter 3 of the NetX Duo User Manual for more information on IPv6 address configuration for more details.

Finally it creates the DNS Client in line 221, and add an IPv6 DNS servers, and a second DNS server limited to IPv4 communication in lines 240 and 252. Note that the DNS Client has its own packet pool which is created when the application calls *nx_dns_create* and which the DNS Client uses to create and send DNS messages. The packet payload and pool size are configuration parameters in *nxd_dns.h* and described later in this chapter.

The remainder of the example program is using the DNS Client services to make DNS queries. Host IP address lookups are performed on lines 267 and 279. Reverse lookups (host name from IP address) are performed on lines 309 and 326.

In this demo, if the DNS Client needs to communicate with the DNS server over IPv6, the USE_IPV6 conditional must be defined in line 19. The DNS Client can look up both IPv4 and IPv6 host addresses, but it does so using IPv6 packets to the DNS Server. Also NetX Duo must be enabled with IPv6 to do so. The use of the conditional `FEATURE_NX_IPV6` prevents the DNS Client application from using IPv6 specific components in NetX Duo inappropriately (e.g. only if IPv6 is enabled in NetX Duo).

The legacy DNS services ***nx_dns_host_by_name_get*** and ***nx_dns_host_by_address_get*** are also demonstrated in lines 289 and 337 respectively, but are limited to IPv4 communication.

Note this demo is created with a 'ram' driver declared on line 40 and which is distributed with the NetX Duo source code. To actually run the DNS Client the application must supply a physical network driver to transmit and receive packets from the DNS server.

```
1    /* This is a small demo of DNS Client for the high-performance NetX Duo TCP/IP
stack.  */
2
3    #include   "tx_api.h"
4    #include   "nx_api.h"
5    #include   "nxd_dns.h"
6
7    #define     DEMO_STACK_SIZE         4096
8
9
10   /* Define the ThreadX and NetX object control blocks...  */
11
12   TX_THREAD               client_thread;
13   NX_PACKET_POOL          client_pool;
14   NX_IP                   client_ip;
15   NX_DNS                  client_dns;
16
17   UINT                    error_counter = 0;
18
19   #define USE_IPV6
20
21   /* If IPv6 is not enabled in NetX Duo, do not allow DNS Client to try using IPv6
*/
22   #ifndef FEATURE_NX_IPV6
23   #undef USE_IPV6
24   #endif
25
26
27   /* Verify NetX Duo version.  */
28   #if (((__NETXDUO_MAJOR_VERSION__  >= 5) && (__NETXDUO_MINOR_VERSION__  >= 6)))
29   #define  MULTIHOME_NETXDUO
30   #endif  /* NETXDUO VERSION check */
31
32   #define CLIENT_ADDRESS      IP_ADDRESS(192,2,2,66)
33   #define DNS_SERVER_ADDRESS  IP_ADDRESS(192,2,2,1)
34
35   /* Define thread prototypes.  */
36
37   void    thread_client_entry(ULONG thread_input);
38
39   /***** Substitute your ethernet driver entry function here *********/
40   VOID    _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
41
42
43   /* Define main entry point.  */
44
45   int main()
46   {
47
48       /* Enter the ThreadX kernel.  */
49       tx_kernel_enter();
```

```
50  }
51
52
53  /* Define what the initial system looks like.  */
54
55  void    tx_application_define(void *first_unused_memory)
56  {
57
58  CHAR    *pointer;
59  UINT    status;
60
61
62      /* Setup the working pointer.  */
63      pointer =  (CHAR *) first_unused_memory;
64
65      /* Create the main thread.  */
66      tx_thread_create(&client_thread, "Client thread", thread_client_entry, 0,
67              pointer, DEMO_STACK_SIZE, 4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
68
69      pointer =  pointer + DEMO_STACK_SIZE;
70
71      /* Initialize the NetX system.  */
72      nx_system_initialize();
73
74      /* Create the packet pool for the DNS Client IP instance to receive packets
           and handle  packet traffic for NetX Duo processes. */
75
76      status =  nx_packet_pool_create(&client_pool, "DNS Client Packet Pool",
                                        1024, pointer, 32000);
77
78      pointer = pointer + 32000;
79
80      /* Check for pool creation error.  */
81      if (status)
82      {
83
84          error_counter++;
85          return;
86       }
87
88      /* Create an IP instance for the DNS Client.  */
89      status = nx_ip_create(&client_ip, "DNS Client IP Instance", CLIENT_ADDRESS,
                              0xFFFFFF00UL, &client_pool,
90                           _nx_ram_network_driver, pointer, 2048, 1);
91
92      pointer =  pointer + 2048;
93
94      /* Check for IP create errors.  */
95      if (status)
96      {
97
98          error_counter++;
99          return;
100      }
101
102     /* Enable ARP and supply ARP cache memory for the DNS Client IP.  */
103     status =  nx_arp_enable(&client_ip, (void *) pointer, 1024);
104     pointer = pointer + 1024;
105
106     /* Check for ARP enable errors.  */
107     if (status)
108     {
109
110         error_counter++;
111         return;
112      }
113
114     /* Enable UDP traffic because DNS is a UDP based protocol.  */
115     status =  nx_udp_enable(&client_ip);
116
117     /* Check for UDP enable errors.  */
118     if (status)
119     {
120
121         error_counter++;
122         return;
123      }
124 }
125
126 #define BUFFER_SIZE 200
```

```
127
128 /* Define the Client thread.  */
129
130 void    thread_client_entry(ULONG thread_input)
131 {
132
133 UCHAR           host_name_buffer[200];
134 UINT            status;
135 ULONG           host_ip_address;
136 #ifdef FEATURE_NX_IPV6
137 NXD_ADDRESS     host_ipduo_address;
138 NXD_ADDRESS     client_ipv6_address;
139 NXD_ADDRESS     dns_ipv6_server_address;
140 NXD_ADDRESS     test_ipduo_server_address;
141 #ifdef MULTIHOME_NETXDUO
142 UINT            iface_index, address_index;
143 #endif
144 #endif
145
146
147     /* Give NetX Duo IP task a chance to get initialized . */
148     tx_thread_sleep(100);
149
150 #ifdef USE_IPV6
151
152     /* Make the DNS Client IPv6 enabled. */
153     status = nxd_ipv6_enable(&client_ip);
154
155     /* Check for enable errors.  */
156     if (status)
157     {
158
159         error_counter++;
160         return;
161      }
162     status = nxd_icmp_enable(&client_ip);
163
164     /* Check for enable errors.  */
165     if (status)
166     {
167
168         error_counter++;
169         return;
170     }
171
172     client_ipv6_address.nxd_ip_address.v6[3] = 0x101;
173     client_ipv6_address.nxd_ip_address.v6[2] = 0x0;
174     client_ipv6_address.nxd_ip_address.v6[1] = 0x0000f101;
175     client_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
176     client_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6;
177
178
179      /* Set the link local address with the host MAC address. */
180 #ifdef MULTIHOME_NETXDUO
181     /* Set the primary interface for our DNS IPv6 addresses. */
182     iface_index = 0;
183
184     /* This assumes we are using the primary network interface (index 0). */
185     status = nxd_ipv6_address_set(&client_ip, iface_index, NX_NULL, 10,
                                      &address_index);
186 #else
187     status = nxd_ipv6_linklocal_address_set(&client_ip, NULL);
188 #endif  /* MULTIHOME_NETXDUO */
189
190     /* Check for link local address set error.  */
191     if (status)
192     {
193
194         error_counter++;
195         return;
196      }
197
198     /* Set the host global IP address. We are assuming a 64
199        bit prefix here but this can be any value (< 128). */
200 #ifdef MULTIHOME_NETXDUO
201     status = nxd_ipv6_address_set(&client_ip, iface_index, &client_ipv6_address,
                                      64, &address_index);
202 #else
203     status = nxd_ipv6_global_address_set(&client_ip, &client_ipv6_address, 64);
204 #endif  /* MULTIHOME_NETXDUO */
205
```

```
206      /* Check for global address set error.  */
207      if (status)
208      {
209
210          error_counter++;
211          return;
212       }
213
214      /* Wait while NetX Duo validates the link local and global address. */
215      tx_thread_sleep(500);
216  #endif
217
218      /* Create a DNS instance for the Client.  Note this function will create
219         the DNS Client packet pool for creating DNS message packets intended
220         for querying its DNS server. */
221      status =  nx_dns_create(&client_dns, &client_ip, (UCHAR *)"DNS Client");
222
223      /* Check for DNS create error.  */
224      if (status)
225      {
226
227          error_counter++;
228          return;
229       }
230
231  #ifdef USE_IPV6
232
233      /* Add an IPv6 DNS server to the DNS client.  */
234      dns_ipv6_server_address.nxd_ip_address.v6[3] = 0x106;
235      dns_ipv6_server_address.nxd_ip_address.v6[2] = 0x0;
236      dns_ipv6_server_address.nxd_ip_address.v6[1] = 0x0000f101;
237      dns_ipv6_server_address.nxd_ip_address.v6[0] = 0x20010db8;
238      dns_ipv6_server_address.nxd_ip_version = NX_IP_VERSION_V6;
239
240      status = nxd_dns_server_add(&client_dns, &dns_ipv6_server_address);
241
242      /* Check for DNS add server error.  */
243      if (status)
244      {
245
246          error_counter++;
247          return;
248       }
249  #else
250
251      /* Add an IPv4 server address to the Client list. */
252      status = nx_dns_server_add(&client_dns, DNS_SERVER_ADDRESS);
253
254      /* Check for DNS add server error.  */
255      if (status)
256      {
257
258          error_counter++;
259          return;
260       }
261  #endif
262
263  #ifdef USE_IPV6
264
265      /* Send a DNS Client name query. Indicate the Client expects an IPv6 address
             (containing an AAAA record). The DNS Client will send this query
266           over IPv6 to its DNS server. */
267      status = nxd_dns_host_by_name_get(&client_dns,
                     (UCHAR *) "www.luxembourg.ipv6ft.org",
                      &host_ipduo_address, 400, NX_IP_VERSION_V6);
268
269      /* Check for DNS query error.  */
270      if (status != NX_SUCCESS)
271      {
272          error_counter++;
273      }
274  #endif
275
276  #ifdef FEATURE_NX_IPV6
277      /* Send a DNS Client name query. Indicate the Client expects an IPv4 address
             (containing an A record). If the DNS client is has an IPv6 DNS server it
278          will send this query over IPv6; otherwise it will be sent over IPv4. */
279      status = nxd_dns_host_by_name_get(&client_dns, (UCHAR *)"www.tahi.org",
                                          &host_ipduo_address, 400, NX_IP_VERSION_V4);
280
281      /* Check for DNS add server error.  */
```

```
282      if (status != NX_SUCCESS)
283      {
284          error_counter++;
285      }
286  #endif
287
288      /* Look up IP address over IPv4. */
289      status = nx_dns_host_by_name_get(&client_dns, (UCHAR *)"www.tahi.org",
&host_ip_address, 400);
290
291      /* Check for DNS add server error.  */
292      if (status != NX_SUCCESS)
293      {
294          error_counter++;
295      }
296
297  #ifdef USE_IPV6
298
299      /* Look up a host name from an IPv6 address (reverse lookup). */
300
301      /* Create an IPv6 address for a reverse lookup. */
302      test_ipduo_server_address.nxd_ip_version = NX_IP_VERSION_V6;
303      test_ipduo_server_address.nxd_ip_address.v6[3] = 0x99;
304      test_ipduo_server_address.nxd_ip_address.v6[2] = 0x0;
305      test_ipduo_server_address.nxd_ip_address.v6[1] = 0x0000f101;
306      test_ipduo_server_address.nxd_ip_address.v6[0] = 0x20010db8;
307
308      /* This will be sent over IPv6 to the DNS server who should return a PTR
            record if it can find the information. */
309      status = nxd_dns_host_by_address_get(&client_dns,
                                    &test_ipduo_server_address,
                                    &host_name_buffer[0], BUFFER_SIZE, 450);
310
311      /* Check for DNS error.  */
312      if (status != NX_SUCCESS)
313      {
314          error_counter++;
315      }
316  #endif
317
318  #ifdef FEATURE_NX_IPV6
319
320      /* Create an IPv4 address for the reverse lookup. If the DNS client is IPv6
           enabled, it will send this over IPv6 to the DNS server; otherwise
321         it will send it over IPv4. In either case the respective server will
322         return a PTR record if it has the information. */
323      test_ipduo_server_address.nxd_ip_version = NX_IP_VERSION_V4;
324      test_ipduo_server_address.nxd_ip_address.v4 = 0xc0020203;
325
326      status = nxd_dns_host_by_address_get(&client_dns,
                            &test_ipduo_server_address, &host_name_buffer[0],
                            BUFFER_SIZE, 450);
327
328       /* Check for DNS query error.  */
329       if (status != NX_SUCCESS)
330       {
331           error_counter++;
332       }
333  #endif
334
335       /* Look up host name over IPv4. */
336       host_ip_address = 0xc0020293;
337       status = nx_dns_host_by_address_get(&client_dns, host_ip_address,
                                    &host_name_buffer[0], BUFFER_SIZE, 450);
338
339       /* Check for DNS query error.  */
340       if (status != NX_SUCCESS)
341       {
342           error_counter++;
343       }
344
345      /* Shutting down...*/
346
347      /* Terminate the DNS Client thread. */
348      status = nx_dns_delete(&client_dns);
349
350      return;
351  }
352
353
```

Figure 1.1 Example application using NetX Duo DNS Client

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
NX_DISABLE_ERROR_CHECKING	Defined, this option removes the basic DNS error checking. It it typically set after the application has been debugged.
NX_DNS_TYPE_OF_SERVICE	Type of service required for the DNS UDP requests. By default, this value is defined as NX_IP_NORMAL for normal IP packet service.
NX_DNS_FRAGMENT_OPTION	Fragment enable for DNS UDP requests. By default, this value is NX_DONT_FRAGMENT.
NX_DNS_TIME_TO_LIVE	Specifies the maximum number of routers a packet can pass before it is discarded. The default value is 0x80.
NX_DNS_MAX_SERVERS	Specifies the maximum number of DNS Servers in the Client server list.
NX_DNS_MESSAGE_MAX	The maximum packet payload size for sending DNS queries. The default value is 512, the limit for DNS queries on IPv4 networks.
NX_DNS_PACKET_PAYLOAD	Size of the Client packet payload which includes the Ethernet, IP, and UDP headers plus the maximum DNS message size specified by NX_DNS_MESSAGE_MAX, and is 4 byte aligned.
NX_DNS_PACKET_POOL_SIZE	Size of the Client packet pool for

sending DNS queries. The default value is large enough for 6 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.

NX_DNS_IP_GATEWAY_SERVER If defined, the DNS Client sets the Client IP gateway as the Client's primary DNS server. The default value is disabled.

NX_DNS_CLIENT_IP_GATEWAY_ADDRESS

This sets IP (version 4) address of the DNS Client IP instance gateway. Only necessary if the NX_DNS_IP_GATEWAY_SERVER option is enabled and the gate IP address is the primary DNS server.

NX_ DNS_PACKET_ALLOCATE_TIMEOUT

This sets the timeout option for allocating a packet from the DNS client packet pool in timer ticks. The default value is 200.

NX_DNS_CLIENT_USER_CREATE_PACKET_POOL

This enables the DNS Client to let the host 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* .

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_create
Create a DNS Client instance

nx_dns_delete
Delete a DNS Client instance

nx_dns_packet_pool_set
Set the DNS Client packet pool

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_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_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_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 make certain the DNS Client host is capable of handling a 512 byte UDP message, not including the UDP, IP and Ethernet headers.

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

NX_SUCCESS	(0x00)	Successful DNS create.
NX_DNS_ERROR	(0xA0)	DNS create error.
NX_PTR_ERROR	(0x16)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(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.  */
```

See Also

nx_dns_delete, nx_dns_host_by_address_get, nx_dns_host_by_name_get, nx_dns_server_add, nx_dns_server_remove

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 messages, so the packet payload should be no less than NX_DNS_PACKET_PAYLOAD defined in *nxd_dns.h.* When the DNS Client is deleted, the packet pool is deleted with 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

dns_ptr	Pointer to previously created DNS Client instance.
pool_ptr	Pointer to previously created packet pool

Return Values

NX_SUCCESS	(0x00)	Successful completion.
NX_NOT_ENABLED	(0x14)	Client not configured for this option
NX_PTR_ERROR	(0x16)	Invalid IP or DNS Client pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
NXD_DNS my_dns;
NX_PACKET_POOL client_pool;

/* Create the DNS Client.  */
status =  nx_dns_create(&my_dns);

/* Create a packet pool for the DNS Client.  */
status =  nx_ packet_pool_create(&client_pool, "DNS Client Packet Pool",
              NX_DNS_PACKET_PAYLOAD, stack_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.  */
```

See Also

nx_dns_create, nx_dns_delete, nx_dns_host_by_address_get,
nx_dns_host_by_name_get, nx_dns_server_add, nx_dns_server_remove

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.

Input Parameters

dns_ptr	Pointer to previously created DNS Client instance.

Return Values

NX_SUCCESS	(0x00)	Successful DNS Client delete.
NX_DNS_ERROR	(0xA0)	Error during DNS Client delete.
NX_PTR_ERROR	(0x16)	Invalid IP or DNS Client pointer.
NX_CALLER_ERROR	(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.  */
```

See Also

nx_dns_create, nx_dns_host_by_address_get, nx_dns_host_by_name_get, nx_dns_server_add, nx_dns_server_remove

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 through 0xFFFFFFFE)
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 (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the DNS resolution.

Return Values

NX_SUCCESS	(0x00)	Successful DNS resolution.
NX_DNS_ERROR	(0xA0)	Internal DNS error.
NX_DNS_FAILED	(0xA3)	Unable to resolve address.

NX_PTR_ERROR	(0x16)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
UCHAR  resolved_name[200];

/* Get the name associated with IP address 130.191.229.14.  */
status =  nx_dns_host_by_address_get(&my_dns, IP_ADDRESS(130,191,229,14),
                      resolved_name, sizeof(resolved_name), 4000);

/* If status is NX_SUCCESS the name associated with the IP address
   “sdsu.edu” can be found in the “resolved_name” string.  */
```

See Also

nxd_dns_host_by_address_get, nx_dns_host_by_name_get, nxd_dns_host_by_name_get

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 0xFFFFFFFE) **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 (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the DNS resolution.

Return Values

NX_SUCCESS	(0x00)	Successful DNS resolution
NX_DNS_NO_SERVER	(0xA1)	No client DNS servers found
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DNS_PARAM_ERROR		

(0xA8) Invalid non pointer input

Allowed From

Threads

Example

```
UCHAR   resolved_name[200];
NXD_ADDRESS host_address;

host_address.nxd_ip_version = NX_IP_VERISON_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 theinput host_address.  */
status =  nxd_dns_host_by_address_get(&my_dns, &host_address,
                      resolved_name, sizeof(resolved_name), 4000);

/* If status is NX_SUCCESS the name associated with the IP address
   “sdsu.edu” can be found in the “resolved_name” string.  */
```

See Also

nx_dns_host_by_address_get, nxd_dns_host_by_name_get
nx_dns_host_by_name_get

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
                  UINT lookup_type);
```

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 0xFFFFFFFE) **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 (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the DNS resolution.

Return Values

NX_SUCCESS	(0x00)	Successful DNS resolution.
NX_DNS_ERROR	(0xA0)	Internal DNS error.
NX_DNS_FAILED	(0xA3)	Unable to resolve name.
NX_PTR_ERROR	(0x16)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
ULONG ip_address;

/* Get the IP address for the name “sdsu.edu”.  */
status =  nx_dns_host_by_name_get(&my_dns, “sdsu.edu”, &ip_address, 4000);

/* If status is NX_SUCCESS the IP address for “sdsu.edu” can be found
   in the “ip_address” variable.  */
```

See Also

nx_dns_host_by_address_get, nxd_dns_host_by_address_get
nxd_dns_host_by_name

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 0xFFFFFFFE) **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 (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the DNS resolution.

Return Values

NX_SUCCESS	(0x00)	Successful DNS resolution
NX_DNS_NO_SERVER	(0xA1)	No client DNS servers found
NX_DNS_FAILED	(0xA3)	Unable to resolve name.
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DNS_PARAM_ERROR	(0xA8)	Invalid non pointer input

Allowed From

Threads

Example

```
NXD_ADDRESS ip_address;

/* Create an AAAA query to obtain the IPv6 address for the host “www.sdsu.edu”.  */
status =  nxd_dns_host_by_name_get(&my_dns, “www.sdsu.edu”, &ip_address, 4000,
NX_IP_VERSION_V6);

/* If status is NX_SUCCESS the IP address for “www.sdsu.edu” can be found
   in the “ip_address” variable.  */

/* Create an A query to obtain the IPv4 address for the host “www.sdsu.edu”.  */
status =  nxd_dns_host_by_name_get(&my_dns, “www.sdsu.edu”, &ip_address, 4000,
NX_IP_VERSION_V4);

/* If status is NX_SUCCESS the IP address for “www.sdsu.edu” can be found
   in the “ip_address” variable.  */
```

See Also

nx_dns_host_by_name_get, nx_dns_host_by_address_get, nxd_dns_host_by_address_get

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 a DNS Server to the Client server list. This is intended to be used with DNS servers who are not IPv6 enabled or for whom the Client only has an IPv4 address. This is a wrapper function for the NetX Duo service *nxd_dns_server_add* which is where the server is actually added to the Client list.

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	Server successfully added
NX_DNS_BAD_ADDRESS_ERROR	(0xA4)	Invalid server address
NX_NO_MORE_ENTRIES	(0x17)	No more DNS Servers Allowed
NX_PTR_ERROR	(0x16)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_IP_ADDRESS_ERROR	(0x21)	Invalid DNS Server IP address

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.  */
```

See Also

nxd_dns_server_add, nx_dns_server_remove, nxd_dns_server_remove

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

dns_ptr	Pointer to DNS control block.
server_address	Pointer to the NXD_ADDRESS containing the server IP address of DNS Server.

Return Values

NX_SUCCESS	(0x00)	Server successfully added
NX_NO_MORE_ENTRIES	(0x17)	Client Server list full
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DNS_PARAM_ERROR	(0xA8)	Invalid non pointer input

Allowed From

Threads

Example

```
NXD_ADDRESS server_address;

server_address.nxd_ip_version = NX_IP_VERISON_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.  */
```

See Also

nx_dns_server_add, nx_dns_server_remove, nxd_dns_server_remove

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 a DNS Server from the Client list. This only supports IPv4 addresses. It is a wrapper function for the NetX Duo Client service *nxd_dns_server_remove* which is actually where the server is removed from the list.

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	Successful DNS Server remove
NX_DNS_ERROR	(0xA0)	Internal DNS error.
NX_PTR_ERROR	(0x16)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_IP_ADDRESS_ERROR	(0x21)	Invalid DNS Server IP address

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.  */
```

See Also

nxd_dns_server_remove, nx_dns_server_add, nxd_dns_server_add

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.

Input Parameters

dns_ptr	Pointer to DNS control block.
server_address	Pointer to DNS Server NXD_ADDRESS data containing server IP address.

Return Values

NX_SUCCESS	(0x00)	Server successfully removed
NX_DNS_SERVER_NOT_FOUND	(0xA9)	Server not in Client list
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DNS_PARAM_ERROR	(0xA8)	Invalid non pointer input

Allowed From

Threads

Example

```
NXD_ADDRESS server_address;

server_address.nxd_ip_version = NX_IP_VERISON_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_DDRESS);

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

See Also

nx_dns_server_remove, nx_dns_server_add, nxd_dns_server_add