



DNS (Domain Name System) Client

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

©2002-2016 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, 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 5.9

Contents

Chapter 1 Introduction to the NetX DNS Client.....	4
DNS Client Setup.....	4
DNS Messages	5
Extended DNS Resource Record Types	6
DNS Cache.....	7
DNS Client Limitations	8
DNS RFCs	8
Chapter 2 Installation and Use of NetX DNS Client	9
Product Distribution.....	9
DNS Client Installation.....	9
Using the DNS Client	9
Small Example System for DNS Client.....	10
Configuration Options	20
Chapter 3 Description of NetX DNS Client Services	23
nx_dns_authority_zone_start_get	25
nx_dns_cache_initialize.....	29
nx_dns_cache_notify_clear.....	30
nx_dns_cache_notify_set.....	31
nx_dns_cname_get.....	32
nx_dns_create	34
nx_dns_delete	35
nx_dns_domain_name_server_get.....	36
nx_dns_domain_mail_exchange_get.....	39
nx_dns_domain_service_get.....	42
nx_dns_get_serverlist_size	45
nx_dns_info_by_name_get	46
nx_dns_ipv4_address_by_name_get	48
nx_dns_host_by_address_get	50
nx_dns_host_by_name_get.....	52
nx_dns_host_text_get	54
nx_dns_packet_pool_set.....	56
nx_dns_server_add	58
nx_dns_server_get	59
nx_dns_server_remove	60
nx_dns_server_remove_all	62

Chapter 1

Introduction to the NetX 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 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.

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 `nx_dns_server_add` service.

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. Please refer to the NetX 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` 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 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 DNS Client can perform IPv4 address lookups (type A) by calling *nx_dns_host_by_name_get* or *nx_dns_ipv4_address_by_name_get*. The DNS Client can perform reverse lookups of IP addresses (type PTR queries) to obtain web host names using *nx_dns_host_by_address_get*.

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 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 DNS Client also supports the following record type queries:

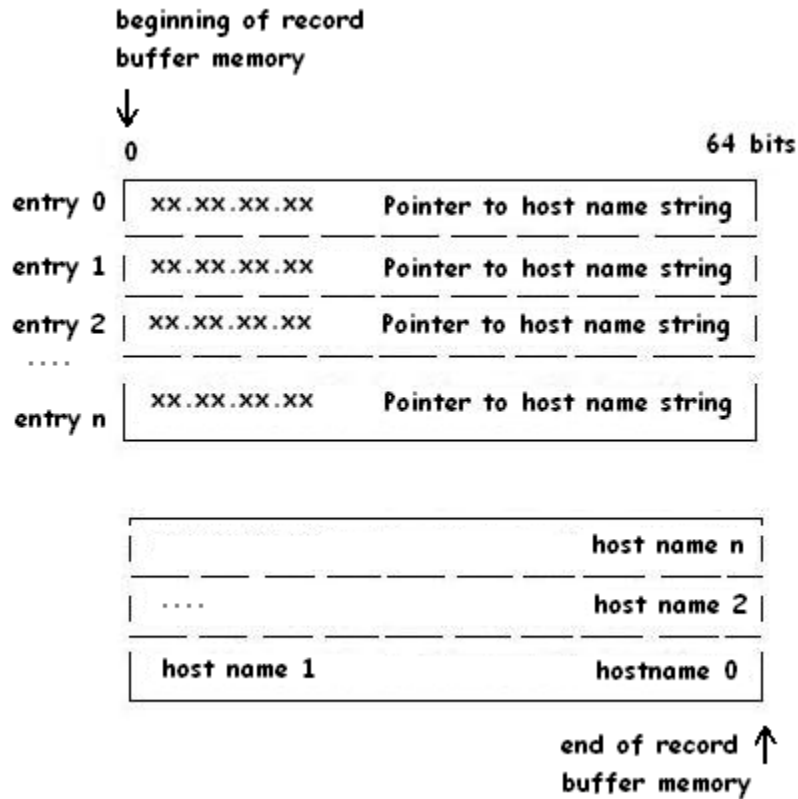
CNAME	contains the canonical name for an alias
TXT	contains a text string
NS	contains an authoritative name server
SOA	contains the start of a zone of authority
MX	used for mail exchange
SRV	contains information on the service offered by the domain

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

For those queries whose record types have variable data length, such as NS records whose host names are of variable length, NetX 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 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 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 DNS Client does not use data from authoritative answers to forward additional DNS queries to other DNS servers.

DNS RFCs

NetX 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)

Chapter 2

Installation and Use of NetX DNS Client

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

Product Distribution

NetX 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:

<code>nx_dns.h</code>	Header file for NetX DNS Client
<code>nx_dns.c</code>	C Source file for NetX DNS Client
<code>nx_dns.pdf</code>	PDF description of NetX DNS Client

DNS Client Installation

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

Using the DNS Client

Using NetX DNS Client is easy. Basically, the application code must include *nx_dns.h* after it includes *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX, respectively. Once *nx_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 *nx_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 DNS.

Note that since DNS utilizes NetX UDP services, UDP must be enabled with the *nx_udp_enable* call prior to using DNS.

Small Example System for DNS Client

In the example DNS application program provided in this section, *nx_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 21-23. 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 71-91. 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 *nx_dns.h* and described elsewhere in this chapter.

Another packet pool is created in lines 93-105 for the Client IP instance which is used for internal NetX operations. Next the IP instance is created using the *nx_ip_create* call in line 107-119. 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 122 and 134 respectively.

Note this demo uses the 'ram' driver declared on line 37 and used in the *nx_ip_create* call. This ram driver is distributed with the NetX 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 on lines 176-187, initializes the cache on lines 189-200, and sets the packet pool previously created to the DNS Client instance on lines 202-217. It then adds an IPv4 DNS server on lines 220-229.

The remainder of the example program uses the DNS Client services to make DNS queries. Host IP address lookups are performed on lines 240 and 262. The difference between these two services, *nx_dns_host_by_name_get* and *nx_dns_ipv4_address_by_name_get*, is that the former only saves one IP address, while the latter saves multiple addresses if DNS Server replied.

Reverse lookups (host name from IP address) are performed on lines 354 (*nx_dns_host_by_address_get*).

Two more services for DNS lookups, CNAME and TXT, are demonstrated on lines 375 and 420 respectively, to discover CNAME and TXT for the input domain name. NetX DNS Client as similar services for other record types, e.g. NS, MX, SRV and SOA. See Chapter 3 for detailed descriptions of all record type lookups available in NetX DNS Client.

When the DNS Client is deleted on line 594, 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 "nx_dns.h"
7
8
9 #define DEMO_STACK_SIZE 4096
10
11 #define NX_PACKET_PAYLOAD 1536
12 #define NX_PACKET_POOL_SIZE 30 * NX_PACKET_PAYLOAD
13 #define LOCAL_CACHE_SIZE 2048
14
15 /* Define the ThreadX and NetX object control blocks... */
16
17 NX_DNS client_dns;
18 TX_THREAD client_thread;
19 NX_IP client_ip;
20 NX_PACKET_POOL main_pool;
21 #ifdef NX_DNS_CLIENT_USER_CREATE_PACKET_POOL
22 NX_PACKET_POOL client_pool;
23 #endif
24 UCHAR local_cache[LOCAL_CACHE_SIZE];
25
26 UINT error_counter = 0;
27
28
29 #define CLIENT_ADDRESS IP_ADDRESS(192,168,0,11)
30 #define DNS_SERVER_ADDRESS IP_ADDRESS(192,168,0,1)
31
32 /* Define thread prototypes. */
33
34 void thread_client_entry(ULONG thread_input);
35
36 /***** Substitute your ethernet driver entry function here *****/
37 extern VOID _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
38
39
40 /* Define main entry point. */
41
42 int main()
43 {
44     /* Enter the ThreadX kernel. */
45     tx_kernel_enter();
46 }
47
48
49
50 /* Define what the initial system looks like. */
51
52 void tx_application_define(void *first_unused_memory)
53 {
54     CHAR *pointer;
55     UINT status;
56
57
58     /* Setup the working pointer. */
59     pointer = (CHAR *) first_unused_memory;
60
61

```

```

62  /* Create the main thread. */
63  tx_thread_create(&client_thread, "Client thread", thread_client_entry, 0,
64  pointer, DEMO_STACK_SIZE, 4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
65
66  pointer = pointer + DEMO_STACK_SIZE;
67
68  /* Initialize the NetX system. */
69  nx_system_initialize();
70
71 #ifdef NX_DNS_CLIENT_USER_CREATE_PACKET_POOL
72
73  /* Create the packet pool for the DNS Client to send packets.
74
75  If the DNS Client is configured for letting the host application create
76  the DNS packet pool, (see NX_DNS_CLIENT_USER_CREATE_PACKET_POOL option),
see
77  nx_dns_create() for guidelines on packet payload size and pool size.
78  packet traffic for NetX processes.
79  */
80  status = nx_packet_pool_create(&client_pool, "DNS Client Packet Pool",
NX_DNS_PACKET_PAYLOAD, pointer, NX_DNS_PACKET_POOL_SIZE);
81
82  pointer = pointer + NX_DNS_PACKET_POOL_SIZE;
83
84  /* Check for pool creation error. */
85  if (status)
86  {
87
88      error_counter++;
89      return;
90  }
91 #endif
92
93  /* Create the packet pool which the IP task will use to send packets. Also
available to the host
94  application to send packet. */
95  status = nx_packet_pool_create(&main_pool, "Main Packet Pool",
NX_PACKET_PAYLOAD, pointer, NX_PACKET_POOL_SIZE);
96
97  pointer = pointer + NX_PACKET_POOL_SIZE;
98
99  /* Check for pool creation error. */
100  if (status)
101  {
102
103      error_counter++;
104      return;
105  }
106
107  /* Create an IP instance for the DNS Client. */
108  status = nx_ip_create(&client_ip, "DNS Client IP Instance", CLIENT_ADDRESS,
0xFFFFF00UL,
109  &main_pool, _nx_ram_network_driver, pointer, 2048, 1);
110
111  pointer = pointer + 2048;
112
113  /* Check for IP create errors. */
114  if (status)
115  {
116
117      error_counter++;
118      return;
119  }
120
121  /* Enable ARP and supply ARP cache memory for the DNS Client IP. */
122  status = nx_arp_enable(&client_ip, (void *) pointer, 1024);
123  pointer = pointer + 1024;
124
125  /* Check for ARP enable errors. */
126  if (status)
127  {
128
129      error_counter++;
130      return;
131  }
132
133  /* Enable UDP traffic because DNS is a UDP based protocol. */
134  status = nx_udp_enable(&client_ip);
135
136  /* Check for UDP enable errors. */
137  if (status)

```

```

138     {
139
140         error_counter++;
141         return;
142     }
143 }
144
145 #define BUFFER_SIZE    200
146 #define RECORD_COUNT    10
147
148 /* Define the Client thread. */
149
150 void    thread_client_entry(ULONG thread_input)
151 {
152
153     UCHAR        record_buffer[200];
154     UINT          record_count;
155     UINT          status;
156     ULONG         host_ip_address;
157     UINT          i;
158     ULONG         *ipv4_address_ptr[RECORD_COUNT];
159 #ifdef NX_DNS_ENABLE_EXTENDED_RR_TYPES
160     NX_DNS_NS_ENTRY *nx_dns_ns_entry_ptr[RECORD_COUNT];
161     NX_DNS_MX_ENTRY *nx_dns_mx_entry_ptr[RECORD_COUNT];
162     NX_DNS_SRV_ENTRY *nx_dns_srv_entry_ptr[RECORD_COUNT];
163     NX_DNS_SOA_ENTRY *nx_dns_soa_entry_ptr;
164     ULONG         host_address;
165     USHORT        host_port;
166 #endif
167
168     /* Give NetX IP task a chance to get initialized . */
169     tx_thread_sleep(100);
170
171     /* Create a DNS instance for the Client. Note this function will create
172     the DNS Client packet pool for creating DNS message packets intended
173     for querying its DNS server. */
174     status = nx_dns_create(&client_dns, &client_ip, (UCHAR *)"DNS Client");
175
176     /* Check for DNS create error. */
177     if (status)
178     {
179         error_counter++;
180         return;
181     }
182
183 #ifdef NX_DNS_CACHE_ENABLE
184     /* Initialize the cache. */
185     status = nx_dns_cache_initialize(&client_dns, local_cache, LOCAL_CACHE_SIZE);
186
187     /* Check for DNS cache error. */
188     if (status)
189     {
190         error_counter++;
191         return;
192     }
193 #endif
194
195     /* Is the DNS client configured for the host application to create the packet
196     pool? */
197 #ifdef NX_DNS_CLIENT_USER_CREATE_PACKET_POOL
198     /* Yes, use the packet pool created above which has appropriate payload size
199     for DNS messages. */
200     status = nx_dns_packet_pool_set(&client_dns, &client_pool);
201
202     /* Check for set DNS packet pool error. */
203     if (status)
204     {
205         error_counter++;
206         return;
207     }
208 #endif /* NX_DNS_CLIENT_USER_CREATE_PACKET_POOL */

```

```

218
219
220     /* Add an IPv4 server address to the Client list. */
221     status = nx_dns_server_add(&client_dns, DNS_SERVER_ADDRESS);
222
223     /* Check for DNS add server error. */
224     if (status)
225     {
226
227         error_counter++;
228         return;
229     }
230
231
232
233
234
235     /*
236     /*
237     /*
238     /*
239     /*
240     status = nx_dns_host_by_name_get(&client_dns, (UCHAR *) "www.my_example.com",
241     &host_ip_address, 400);
242
243     /* Check for DNS query error. */
244     if (status != NX_SUCCESS)
245     {
246         error_counter++;
247     }
248     else
249     {
250
251         printf("-----\n");
252         printf("Test A: \n");
253         printf("IP address: %lu.%lu.%lu.%lu\n",
254             host_ip_address >> 24,
255             host_ip_address >> 16 & 0xFF,
256             host_ip_address >> 8 & 0xFF,
257             host_ip_address & 0xFF);
258     }
259
260
261     /* Look up IPv4 addresses to record multiple IPv4 addresses in record_buffer
262     and return the IPv4 address count. */
263     status = nx_dns_ipv4_address_by_name_get(&client_dns, (UCHAR
264     *) "www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
265
266     /* Check for DNS query error. */
267     if (status != NX_SUCCESS)
268     {
269         error_counter++;
270     }
271     else
272     {
273
274         printf("-----\n");
275         printf("Test A: ");
276         printf("record_count = %d \n", record_count);
277     }
278
279     /* Get the IPv4 addresses of host. */
280     for(i = 0; i < record_count; i++)
281     {
282         ipv4_address_ptr[i] = (ULONG *) (record_buffer + i * sizeof(ULONG));
283         printf("record %d: IP address: %lu.%lu.%lu.%lu\n", i,
284             *ipv4_address_ptr[i] >> 24,
285             *ipv4_address_ptr[i] >> 16 & 0xFF,
286             *ipv4_address_ptr[i] >> 8 & 0xFF,
287             *ipv4_address_ptr[i] & 0xFF);
288     }
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
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

```

```

291 /*                                     Type A + CNAME response
292 /*                                     Send A type DNS Query to its DNS server and get the IPv4 address.
293 /*
294 /* Look up an IPv4 address over IPv4. */
295 status = nx_dns_host_by_name_get(&client_dns, (UCHAR *)"www.my_example.com",
&host_ip_address, 400);
296
297 /* Check for DNS query error. */
298 if (status != NX_SUCCESS)
299 {
300     error_counter++;
301 }
302
303 else
304 {
305
306     printf("-----\n");
307     printf("Test A + CNAME response: \n");
308     printf("IP address: %lu.%lu.%lu.%lu\n",
309         host_ip_address >> 24,
310         host_ip_address >> 16 & 0xFF,
311         host_ip_address >> 8 & 0xFF,
312         host_ip_address & 0xFF);
313 }
314
315
316 /* Look up IPv4 addresses to record multiple IPv4 addresses in record_buffer
and return the IPv4 address count. */
317 status = nx_dns_ipv4_address_by_name_get(&client_dns, (UCHAR
*)"www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
318
319 /* Check for DNS query error. */
320 if (status != NX_SUCCESS)
321 {
322     error_counter++;
323 }
324
325 else
326 {
327
328     printf("-----\n");
329     printf("Test Test A + CNAME response: ");
330     printf("record_count = %d \n", record_count);
331 }
332
333 /* Get the IPv4 addresses of host. */
334 for(i = 0; i < record_count; i++)
335 {
336     ipv4_address_ptr[i] = (ULONG *)(&record_buffer + i * sizeof(ULONG));
337     printf("record %d: IP address: %lu.%lu.%lu.%lu\n", i,
338         *ipv4_address_ptr[i] >> 24,
339         *ipv4_address_ptr[i] >> 16 & 0xFF,
340         *ipv4_address_ptr[i] >> 8 & 0xFF,
341         *ipv4_address_ptr[i] & 0xFF);
342 }
343
344
345
346 /*                                     Type PTR
347 /*                                     Send PTR type DNS Query to its DNS server and get the host name.
348 /*
349 /*
350
351
352 /* Look up host name over IPv4. */
353 host_ip_address = IP_ADDRESS(74, 125, 71, 106);
354 status = nx_dns_host_by_address_get(&client_dns, host_ip_address,
&record_buffer[0], BUFFER_SIZE, 450);
355
356 /* Check for DNS query error. */
357 if (status != NX_SUCCESS)
358 {
359     error_counter++;
360 }

```

```

361
362     else
363     {
364         printf("-----\n");
365         printf("Test PTR: %s\n", record_buffer);
366     }
367
368 #ifdef NX_DNS_ENABLE_EXTENDED_RR_TYPES
369
370 /*-----
371 /*                                     Type CNAME
372 /*
373 /* Send CNAME type DNS Query to its DNS server and get the canonical name .
374 /*
375 /*-----
376 /* Send CNAME type to record the canonical name of host in record_buffer.
377 /*
378 status = nx_dns_cname_get(&client_dns, (UCHAR *) "www.my_example.com",
379 &record_buffer[0], BUFFER_SIZE, 400);
380
381 /* Check for DNS query error. */
382 if (status != NX_SUCCESS)
383 {
384     error_counter++;
385 }
386
387 else
388 {
389     printf("-----\n");
390     printf("Test CNAME: %s\n", record_buffer);
391 }
392
393 /*-----
394 /*                                     Type TXT
395 /*
396 /* Send TXT type DNS Query to its DNS server and get descriptive text.
397 /*
398 /*-----
399 /* Send TXT type to record the descriptive test of host in record_buffer.
400 /*
401 status = nx_dns_host_text_get(&client_dns, (UCHAR *) "www.my_example.com",
402 &record_buffer[0], BUFFER_SIZE, 400);
403
404 /* check for DNS query error. */
405 if (status != NX_SUCCESS)
406 {
407     error_counter++;
408 }
409
410 else
411 {
412     printf("-----\n");
413     printf("Test TXT: %s\n", record_buffer);
414 }
415
416 /*-----
417 /*                                     Type NS
418 /*
419 /* Send NS type DNS Query to its DNS server and get the domain name server.
420 /*
421 /*-----
422 /* Send NS type to record multiple name servers in record_buffer and return
423 the name server count.
424 If the DNS response includes the IPv4 addresses of name server, record it
425 similarly in record_buffer. */
426 status = nx_dns_domain_name_server_get(&client_dns, (UCHAR
427 *) "www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
428
429 /* check for DNS query error. */

```



```

423     if (status != NX_SUCCESS)
424     {
425         error_counter++;
426     }
427
428     else
429     {
430
431         printf("-----\n");
432         printf("Test NS: ");
433         printf("record_count = %d \n", record_count);
434     }
435
436     /* Get the name server. */
437     for(i=0; i< record_count; i++)
438     {
439         nx_dns_ns_entry_ptr[i] = (NX_DNS_NS_ENTRY *) (record_buffer + i *
440         sizeof(NX_DNS_NS_ENTRY));
441         printf("record %d: IP address: %d.%d.%d.%d\n", i,
442             nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 24,
443             nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 16 & 0xFF,
444             nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address >> 8 & 0xFF,
445             nx_dns_ns_entry_ptr[i] -> nx_dns_ns_ipv4_address & 0xFF);
446         if(nx_dns_ns_entry_ptr[i] -> nx_dns_ns_hostname_ptr)
447             printf("hostname = %s\n", nx_dns_ns_entry_ptr[i] ->
448             nx_dns_ns_hostname_ptr);
449         else
450             printf("hostname is not set\n");
451     }
452
453     /*
454     /* Send MX type DNS Query to its DNS server and get the domain mail exchange.
455     /*
456     /* Send MX DNS query type to record multiple mail exchanges in record_buffer
457     and return the mail exchange count.
458     If the DNS response includes the IPv4 addresses of mail exchange, record
459     it similarly in record_buffer. */
460     status = nx_dns_domain_mail_exchange_get(&client_dns, (UCHAR
461     *) "www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
462
463     /* Check for DNS query error. */
464     if (status != NX_SUCCESS)
465     {
466         error_counter++;
467     }
468
469     else
470     {
471         printf("-----\n");
472         printf("Test MX: ");
473         printf("record_count = %d \n", record_count);
474     }
475
476     /* Get the mail exchange. */
477     for(i=0; i< record_count; i++)
478     {
479         nx_dns_mx_entry_ptr[i] = (NX_DNS_MX_ENTRY *) (record_buffer + i *
480         sizeof(NX_DNS_MX_ENTRY));
481         printf("record %d: IP address: %d.%d.%d.%d\n", i,
482             nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 24,
483             nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 16 & 0xFF,
484             nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address >> 8 & 0xFF,
485             nx_dns_mx_entry_ptr[i] -> nx_dns_mx_ipv4_address & 0xFF);
486         printf("preference = %d \n", nx_dns_mx_entry_ptr[i] ->
487         nx_dns_mx_preference);
488         if(nx_dns_mx_entry_ptr[i] -> nx_dns_mx_hostname_ptr)
489             printf("hostname = %s\n", nx_dns_mx_entry_ptr[i] ->
490             nx_dns_mx_hostname_ptr);
491         else
492             printf("hostname is not set\n");
493     }

```

```

492
493 /*
494 /* Send SRV type DNS Query to its DNS server and get the location of services.
495
496
497 /* Send SRV DNS query type to record the location of services in
498 record_buffer and return count.
499 If the DNS response includes the IPv4 addresses of service name, record
500 it similarly in record_buffer. */
501 status = nx_dns_domain_service_get(&client_dns, (UCHAR
502 *)"www.my_example.com", &record_buffer[0], BUFFER_SIZE, &record_count, 400);
503
504 /* Check for DNS query error. */
505 if (status != NX_SUCCESS)
506 {
507     error_counter++;
508 }
509 else
510 {
511     printf("-----\n");
512     printf("Test SRV: ");
513     printf("record_count = %d \n", record_count);
514 }
515 /* Get the location of services. */
516 for(i = 0; i < record_count; i++)
517 {
518     nx_dns_srv_entry_ptr[i] = (NX_DNS_SRV_ENTRY *) (record_buffer + i *
519     sizeof(NX_DNS_SRV_ENTRY));
520     printf("record %d: IP address: %d.%d.%d.%d\n", i,
521     nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 24,
522     nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 16 & 0xFF,
523     nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address >> 8 & 0xFF,
524     nx_dns_srv_entry_ptr[i] -> nx_dns_srv_ipv4_address & 0xFF);
525     printf("port number = %d\n", nx_dns_srv_entry_ptr[i] ->
526     nx_dns_srv_port_number);
527     printf("priority = %d\n", nx_dns_srv_entry_ptr[i] ->
528     nx_dns_srv_priority);
529     printf("weight = %d\n", nx_dns_srv_entry_ptr[i] -> nx_dns_srv_weight);
530     if(nx_dns_srv_entry_ptr[i] -> nx_dns_srv_hostname_ptr)
531     printf("hostname = %s\n", nx_dns_srv_entry_ptr[i] ->
532     nx_dns_srv_hostname_ptr);
533     else
534     printf("hostname is not set\n");
535 }
536
537 /* Get the service info, NetX old API.*/
538 status = nx_dns_info_by_name_get(&client_dns, (UCHAR *)"www.my_example.com",
539 &host_address, &host_port, 200);
540
541 /* Check for DNS add server error. */
542 if (status != NX_SUCCESS)
543 {
544     error_counter++;
545 }
546 else
547 {
548     printf("-----\n");
549     printf("Test SRV: ");
550     printf("IP address: %d.%d.%d.%d\n",
551     host_address >> 24,
552     host_address >> 16 & 0xFF,
553     host_address >> 8 & 0xFF,
554     host_address & 0xFF);
555     printf("port number = %d\n", host_port);
556 }
557
558
559 /*
560 /*

```

```

558 /* Send SOA type DNS Query to its DNS server and get zone of start of
559 authority.*/
560 /******
561      /* Send SOA DNS query type to record the zone of start of authority in
562 record_buffer. */
563      status = nx_dns_authority_zone_start_get(&client_dns, (UCHAR
564 *)"www.my_example.com", &record_buffer[0], BUFFER_SIZE, 400);
565
566      /* Check for DNS query error. */
567      if (status != NX_SUCCESS)
568      {
569          error_counter++;
570      }
571
572      /* Get the loc*/
573      nx_dns_soa_entry_ptr = (NX_DNS_SOA_ENTRY *) record_buffer;
574      printf("-----\n");
575      printf("Test SOA: \n");
576      printf("serial = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_serial );
577      printf("refresh = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_refresh );
578      printf("retry = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_retry );
579      printf("expire = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_expire );
580      printf("minum = %d\n", nx_dns_soa_entry_ptr -> nx_dns_soa_minum );
581      if(nx_dns_soa_entry_ptr -> nx_dns_soa_host_mname_ptr)
582          printf("host mname = %s\n", nx_dns_soa_entry_ptr ->
583 nx_dns_soa_host_mname_ptr);
584      else
585          printf("host mame is not set\n");
586      if(nx_dns_soa_entry_ptr -> nx_dns_soa_host_rname_ptr)
587          printf("host rname = %s\n", nx_dns_soa_entry_ptr ->
588 nx_dns_soa_host_rname_ptr);
589      else
590          printf("host rname is not set\n");
591
592 #endif
593
594 /* Shutting down...*/
595
596 /* Terminate the DNS Client thread. */
597 status = nx_dns_delete(&client_dns);
598
599 return;
600 }
601
602
603

```

Configuration Options

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

Define	Meaning
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_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 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.
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 if NX_DNS_CLIENT_USER_CREATE_PACKET_POOL is not defined. 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_MAX_RETRANS_TIMEOUT The maximum retransmission timeout on a DNS query to a specific DNS server. The default value is 64 seconds.

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 in timer ticks. The default value is 200.

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 retrieve multiple DNS server responses off the DNS Client queue until it finds a response that matches the current query. Older packets from previous DNS queries are discarded to prevent the DNS Client socket 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 NetX 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

`nx_dns_host_by_address_get`

Look up a host name from a specified IP address

`nx_dns_host_by_name_get`

Look up the IPv4 address from the specified host name

`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`

*Add a DNS Server at the specified address
to the Client list*

`nx_dns_server_get`

Return the DNS Server in the Client list

`nx_dns_server_remove`

Remove a DNS Server from the Client list

`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);
```

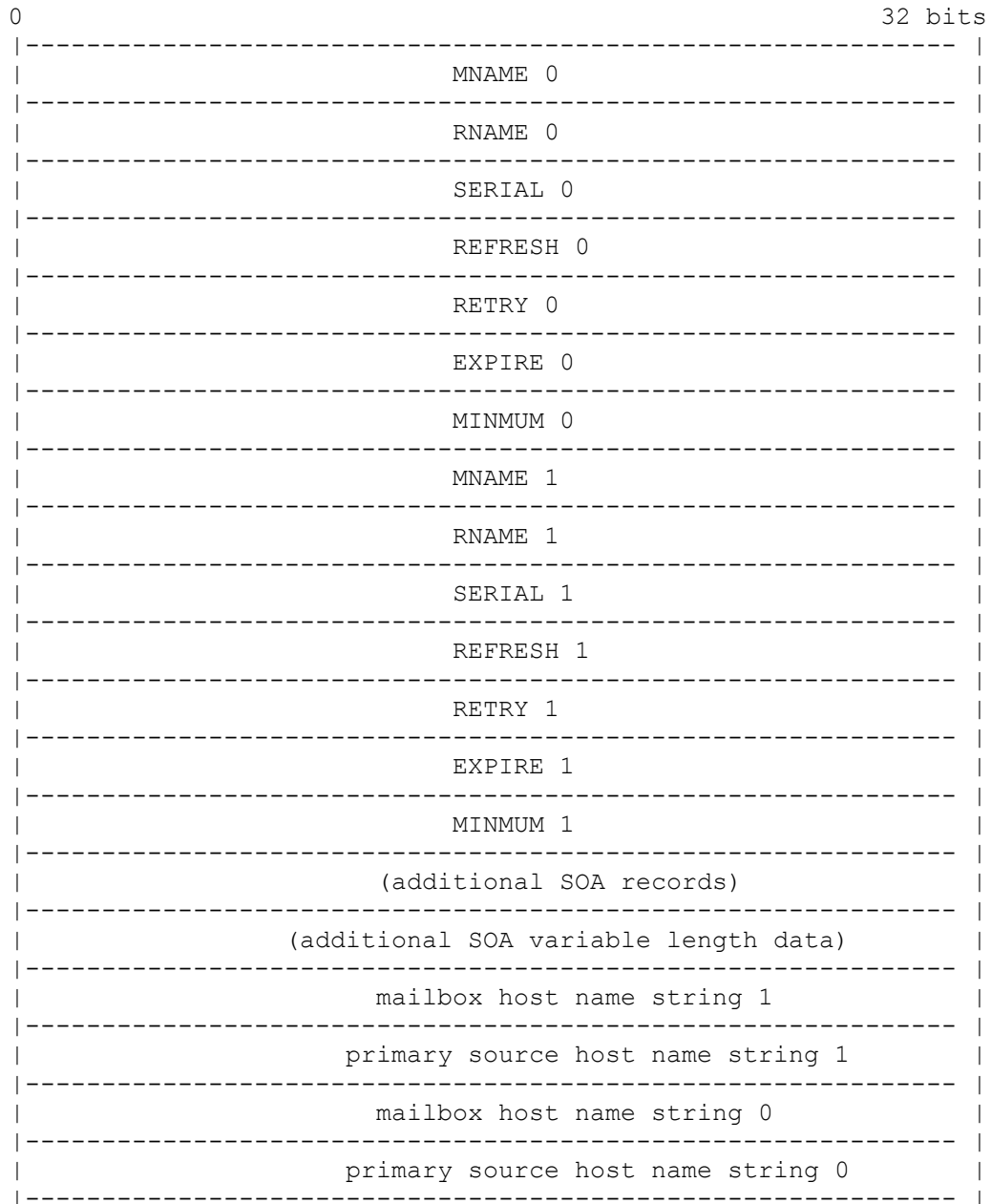
Description

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 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

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

Return Values

NX_SUCCESS	(0x00)	Successfully obtained SOA data
NX_DNS_NO_SERVER	(0xA1)	Client server list is empty
NX_DNS_QUERY_FAILED	(0xA3)	No valid DNS response received
NX_DNS_NEED_MORE_RECORD_BUFFER	(0xB4)	The input buffer is not large enough to hold the minimum data
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

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

dns_ptr	Pointer to DNS control block.
cache_ptr	Pointer to DNS Cache.
cache_size	Size of DNS Cache, in bytes.

Return Values

NX_SUCCESS	(0x00)	DNS Cache successfully initialized
NX_DNS_ERROR	(0xA0)	Cache is not 4-byte aligned.
NX_DNS_PARAM_ERROR	(0xA8)	Invalid DNS ID.
NX_DNS_CACHE_ERROR	(0xB7)	Invalid Cache pointer.
NX_PTR_ERROR	(0x07)	Invalid DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
UCHAR dns_cache [2048];

/* 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

dns_ptr	Pointer to DNS control block.
----------------	-------------------------------

Return Values

NX_SUCCESS	(0x00)	DNS cache notify successfully set
NX_DNS_PARAM_ERROR	(0xA8)	Invalid DNS ID.
NX_PTR_ERROR	(0x07)	Invalid DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

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

dns_ptr	Pointer to DNS control block.
cache_full_notify_cb	The callback function to be invoked when cache become full.

Return Values

NX_SUCCESS	(0x00)	DNS cache notify successfully set
NX_DNS_PARAM_ERROR	(0xA8)	Invalid DNS ID.
NX_PTR_ERROR	(0x07)	Invalid DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

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 *nx_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

NX_SUCCESS	(0x00)	Successfully obtained CNAME data
NX_DNS_NO_SERVER	(0xA1)	Client server list is empty
NX_DNS_QUERY_FAILED	(0xA3)	No valid DNS response received
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

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`. 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.

Input Parameters

<code>dns_ptr</code>	Pointer to DNS Client.
<code>ip_ptr</code>	Pointer to previously created IP instance.
<code>domain_name</code>	Pointer to domain name for DNS instance.

Return Values

NX_SUCCESS	(0x00)	Successful DNS create
NX_DNS_ERROR	(0xA0)	DNS create error
status		Completion status of internal NetX and ThreadX calls
<code>NX_PTR_ERROR</code>	(0x07)	Invalid IP or DNS pointer
<code>NX_CALLER_ERROR</code>	(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

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
status		Completion status of internal NetX and ThreadX calls
NX_PTR_ERROR	(0x07)	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. */
```

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 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

<code>NX_SUCCESS</code>	(0x00)	Successfully obtained NS data
<code>NX_DNS_NO_SERVER</code>	(0xA1)	Client server list is empty
<code>NX_DNS_QUERY_FAILED</code>	(0xA3)	No valid DNS response received
<code>NX_DNS_NEED_MORE_RECORD_BUFFER</code>	(0xB4)	The input buffer is not large enough to hold the minimum data
<code>NX_PTR_ERROR</code>	(0x07)	Invalid IP or DNS pointer
<code>NX_CALLER_ERROR</code>	(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 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

NX_SUCCESS	(0x00)	Successfully obtained MX data
NX_DNS_NO_SERVER	(0xA1)	Client server list is empty
NX_DNS_QUERY_FAILED	(0xA3)	No valid DNS response received
NX_DNS_NEED_MORE_RECORD_BUFFER	(0xB4)	The input buffer is not large enough to hold the minimum data
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer
NX_CALLER_ERROR	(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.aspx.1.www.my_example.com
record 1: IP address: 192.2.2.11
preference = 50
hostname = alt4.aspx.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.aspx.1.www.my_example.com
record 4: IP address: 192.2.2.14
preference = 30
hostname = alt2.aspx.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

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

Return Values

NX_SUCCESS	(0x00)	Successfully obtained SRV data
NX_DNS_NO_SERVER	(0xA1)	Client server list is empty
NX_DNS_QUERY_FAILED	(0xA3)	No valid DNS response received
NX_DNS_NEED_MORE_RECORD_BUFFER	(0xB4)	The input buffer is not large enough to hold the minimum data
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer
NX_CALLER_ERROR	(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.1.www.my_example.com
record 1: IP address: 192.2.2.11
port number = 5222
priority = 5
weight = 0
hostname = xmpp.1.www.my_example.com
record 2: IP address: 192.2.2.12
port number = 5222
priority = 20
weight = 0
hostname = alt1.xmpp.1.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 in the Client list.

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	DNS Server list size successfully returned
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(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

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

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

NX_SUCCESS	(0x00)	Successfully obtained IPv4 data
NX_DNS_NO_SERVER	(0xA1)	Client server list is empty
NX_DNS_QUERY_FAILED		

	(0xA3)	No valid DNS response received
NX_DNS_NEED_MORE_RECORD_BUFFER	(0xB4)	Input buffer not large enough to hold the minimum data
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

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
```

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*.

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	(0x00000001-0xFFFFFFFFE)
TX_WAIT_FOREVER	(0xFFFFFFFFF)

Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a DNS server responds to the request.

Selecting a numeric value (1-0xFFFFFFFFE) 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_TIMEOUT	(0xA2)	Timed out on obtaining DNS mutex
NX_DNS_NO_SERVER	(0xA1)	No DNS Server address specified
NX_DNS_QUERY_FAILED		

	(0xA3)	Received no response to query
NX_DNS_BAD_ADDRESS_ERROR	(0xA4)	Null input address
NX_DNS_PARAM_ERROR		
	(0xA8)	Invalid non pointer input
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_DNS_PARAM_ERROR		
	(0xA8)	Invalid non pointer input

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. */
```

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*.

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

NX_SUCCESS	(0x00)	Successful DNS resolution.
NX_DNS_NO_SERVER	(0xA1)	No DNS Server address specified
NX_DNS_QUERY_FAILED	(0xA3)	Received no response to query
NX_DNS_BAD_ADDRESS_ERROR	(0xA4)	Null input address
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 input

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

```

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

NX_SUCCESS	(0x00)	Successfully TXT string obtained
NX_DNS_NO_SERVER	(0xA1)	Client server list is empty
NX_DNS_QUERY_FAILED	(0xA3)	No valid DNS response received
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_DNS_PARAM_ERROR	(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 be no less than NX_DNS_PACKET_PAYLOAD which includes the IP and UDP headers and is defined in *nx_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 *nx_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	(0x07)	Invalid IP or DNS Client pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
NX_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

NX_SUCCESS	(0x00)	Server successfully added
NX_DNS_BAD_ADDRESS_ERROR	(0xA4)	Null server address input
NX_DNS_DUPLICATE_ENTRY	(0x17)	No more DNS Servers Allowed (list is full)
NX_NO_MORE_ENTRIES	(0x07)	Invalid pointer input
NX_PTR_ERROR	(0xA8)	Invalid non pointer input
NX_DNS_PARAM_ERROR	(0x11)	Invalid caller of this service
NX_CALLER_ERROR		

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. */
```

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 error is returned. The *nx_dns_get_serverlist_size* service may be called first obtain the number of DNS servers in the Client list.

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

NX_SUCCESS	(0x00)	Successful server returned
NX_DNS_SERVER_NOT_FOUND	(0xA9)	Index points to empty slot
NX_DNS_BAD_ADDRESS_ERROR	(0xA4)	Index points to Null address
NX_DNS_PARAM_ERROR	(0xA8)	Index exceeds size of list
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(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. */
```

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.

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	DNS Server successfully removed
NX_DNS_SERVER_NOT_FOUND	(0xA9)	Server not in Client list
NX_DNS_BAD_ADDRESS_ERROR	(0xA4)	Null server address input
status		Completion status of internal NetX and ThreadX calls
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_DNS_BAD_ADDRESS_ERROR	(0xA4)	Null Server address input

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. */
```

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

dns_ptr	Pointer to DNS control block.
----------------	-------------------------------

Return Values

NX_SUCCESS	(0x00)	DNS Servers successfully removed
NX_DNS_ERROR	(0xA0)	Unable to obtain protection mutex to remove servers
NX_PTR_ERROR	(0x07)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(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. */
```