N E T X™

the high-performance real-time implementation
of TCP/IP standards

Domain Name System (DNS)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1051
Revision 5.0

Contents

Chapter 1

Introduction to DNS Client

NetX applications use IP addresses to communicate with all other entities on the network. Before anything can be sent via NetX, the destination's IP address must be known. This can be done statically by defining destination IP addresses inside the application. Alternatively, destination IP addresses may be derived dynamically at run-time. Dynamic IP address definition is accomplished through user interaction or by utilizing the Internet's Domain Name System (DNS).

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. Please refer to the 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. Additional DNS Servers may be specified through the ***nx_dns_server_add*** service.

DNS Names

As mentioned previously, DNS names are hierarchical. The reason for this is manageability. The mapping responsibility for hierarchical names is more easily delegated to different authorities, thereby making the responsibility for the name-to-IP address mapping more manageable.

Hierarchical DNS names consist of one or more labels that are fixed in size at 63 characters and are separated by a period. Each label represents a domain. The hierarchy is represented by reading the name from right-to-left, where the rightmost label represents the top level domain. For example, the DNS name cs.sdsu.edu consists of four domains, the highest level of which is the “edu” domain. The lowest level is the “cs” domain.

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. 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.

DNS Communication

DNS utilizes the UDP protocol on port number 53 to send requests and field responses.

DNS Multi-Thread Support

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.

DNS RFCs

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

Chapter 2

Installation and Use of DNS Client

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

Product Distribution

DNS for NetX 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:

nx_dns.h	Header file for DNS for NetX
nx_dns.c	C Source file for DNS for NetX
nx_dns.pdf	PDF description of DNS for NetX

DNS Installation

In order to use DNS for NetX, the entire distribution mentioned previously should be copied 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 DNS

Using DNS for NetX 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 include *nx_dns.c* in 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

An example of how easy it is to use NetX DNS is described in Figure 1.1 that appears below. In this example, the DNS include file *nx_dns.h* is brought in at line 3. Next, DNS is created in "*my_thread_entry*" at line 92. Note that the DNS control block "*my_dns*" was defined as a global variable at line 9 previously. After successful creation, a DNS Server is specified at line 99. At line 106 the example system attempts to get the IP address for sdsu.edu. If successful, the derived IP address may be used for communication with sdsu.edu.

```
0001 #include   "tx_api.h"
0002 #include   "nx_api.h"
0003 #include   "nx_dns.h"
0004
0005 #define     DEMO_STACK_SIZE         4096
0006 TX_THREAD               my_thread;
0007 NX_PACKET_POOL          my_pool;
0008 NX_IP                   my_ip;
0009 NX_DNS                  my_dns;
0010
0011 /* Define function prototypes.  */
0012
0013 void    my_thread_entry(ULONG thread_input);
0014 void    my_netx_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
0015
0016 /* Define main entry point.  */
0017
0018 int main()
0019 {
0020
0021     /* Enter the ThreadX kernel.  */
0022     tx_kernel_enter();
0023 }
0024
0025
0026 /* Define what the initial system looks like.  */
0027
0028 void    tx_application_define(void *first_unused_memory)
0029 {
0030
0031 CHAR    *pointer;
0032 UINT    status;
0033
0034
0035     /* Setup the working pointer.  */
0036     pointer =  (CHAR *) first_unused_memory;
0037
0038     /* Create “my_thread”.  */
0039      tx_thread_create(&my_thread, "my thread", my_thread_entry, 0,
0040                  pointer, DEMO_STACK_SIZE,
0041                  2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
0042     pointer =  pointer + DEMO_STACK_SIZE;
0043
0044     /* Initialize the NetX system.  */
0045     nx_system_initialize();
0046
0047     /* Create a packet pool.  */
0048     status =  nx_packet_pool_create(&my_pool, "NetX Main Packet Pool",
0049                                          1024, pointer, 64000);
0050     pointer = pointer + 64000;
0051
0052     /* Check for pool creation error.  */
0053     if (status)
0054         error_counter++;
0055
0056     /* Create an IP instance with an IP address of 216.2.3.1. */
0057     status = nx_ip_create(&my_ip,"My NetX IP Instance",
0058             IP_ADDRESS(216,2,3,1), 0xFFFFFF00, &my_pool,
0059             my_netx_driver, pointer, DEMO_STACK_SIZE, 1);
0060     pointer =  pointer + DEMO_STACK_SIZE;
0061
```

```
    /* Check for IP create errors.  */
    if (status)
        error_counter++;

    /* Enable ARP and supply ARP cache memory for my IP Instance.  */
    status =  nx_arp_enable(&my_ip, (void *) pointer, 1024);
    pointer = pointer + 1024;

    /* Check for ARP enable errors.  */
    if (status)
        error_counter++;

    /* Enable UDP.  */
    status =  nx_udp_enable(&my_ip);
    if (status)
        error_counter++;
}


/* Define my thread.  */

void    my_thread_entry(ULONG thread_input)
{

UINT        status;
ULONG       address;
NX_PACKET   *my_packet;


    /* Create a DNS instance for my_ip.  */
    status =  nx_dns_create(&my_dns, &my_ip, “my_name”);

    /* Check for DNS create error.  */
    if (status)
       error_counter++;

    /* Add a DNS server IP address.  */
    status = nx_dns_server_add(&my_dns, IP_ADDRESS(216,2,3,99));

    /* Check for DNS add server error.  */
    if (status)
        error_counter++;

    /* Now attempt to get the IP address for sdsu.edu.  */
    status =  nx_dns_host_by_name_get(&my_dns, “sdsu.edu”,
                                  &address, 0x40000);

    /* Check for DNS status.  */
    if (status == NX_SUCCESS)
    {
        /* Yes, the DNS name resolution was successful! The
           “address” variable contains the IP address of
           sdsu.edu.  */
    }
   else
    {
        /* DNS mapping was not found!  */

    }
}
```

Figure 1.1 Example of DNS use with NetX

Configuration Options

There are several configuration options for building DNS for NetX. 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 used 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 to indicate normal IP packet service. This define can be set by the application prior to inclusion of *nx_dns.h*.
NX_DNS_FRAGMENT_OPTION	Fragment enable for DNS UDP requests. By default, this value is NX_DONT_FRAGMENT to disable DNS UDP fragmenting. This define can be set by the application prior to inclusion of *nx_dns.h*.
NX_DNS_TIME_TO_LIVE	Specifies the number of routers this packet can pass before it is discarded. The default value is set to 0x80, but can be redefined prior to inclusion of *nx_dns.h.*
NX_DNS_MAX_SERVERS	Specifies the maximum number of DNS Servers to examine when trying to resolve a domain name or IP address.
NX_DNS_PACKETS_IN_POOL	Specifies the number packets to

create in the DNS packet pool. The default value is 1, defined in *nx_dns.h*
.

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 instance

nx_dns_delete
Delete a DNS instance

nx_dns_host_by_address_get
Get host name from IP address

nx_dns_host_by_address_get
Get IP address from host name

nx_dns_info_by_name_get,
Get IP address and port from host name

nx_dns_server_add
Add DNS Server IP address

nx_dns_server_remove
Remove DNS Server IP address

nx_dns_server_remove_all
Removes all DNS Servers

nx_dns_create

Create a DNS instance

Prototype

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

Description

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

Important Note: The application must make certain it is capable of handling a 512 byte UDP message, not including the UDP, IP and Ethernet headers.

Input Parameters

dns_ptr	Pointer to DNS control block.
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 instance.  */
status =  nx_dns_create(&my_dns, &my_ip, "My DNS");

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

See Also

nx_dns_delete, nx_dns_host_by_address_get, nx_dns_host_by_name_get, nx_dns_info_by_name_get, nx_dns_server_add, nx_dns_server_remove

nx_dns_delete

Delete a DNS instance

Prototype

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

Description

This service deletes a previously created DNS instance.

Input Parameters

dns_ptr	Pointer to previously created DNS instance.

Return Values

NX_SUCCESS	(0x00)	Successful DNS delete.
NX_DNS_ERROR	(0xA0)	Error during DNS delete.
NX_PTR_ERROR	(0x16)	Invalid IP or DNS pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

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

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

See Also

nx_dns_create, nx_dns_host_by_address_get, nx_dns_host_by_name_get, nx_dns_info_by_name_get, nx_dns_server_add, nx_dns_server_remove, nx_dns_server_remove

nx_dns_host_by_address_get

Get name from 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 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 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

nx_dns_create, nx_dns_delete, nx_dns_host_by_name_get, nx_dns_info_by_name_get, nx_dns_server_add, nx_dns_server_remove, nx_dns_server_remove

nx_dns_host_by_name_get

Get IP address from 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 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_create, nx_dns_delete, nx_dns_host_by_address_get, nx_dns_info_by_name_get, nx_dns_server_add, nx_dns_server_remove, nx_dns_server_remove

nx_dns_info_by_name_get

Get IP address, port from host name

Prototype

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

Description

This service requests IP address and port of the supplied name from one or more DNS Servers previously specified by the application. If successful, the associated IP address and port is returned in the destination pointed to by *host_address_ptr* and *host_port_ptr* respectively.

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
host_port_ptr	Pointer to destination for port
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_TIMEOUT	(0xA2)	Unable to obtain lock on DNS
NX_DNS_NO_SERVER	(0xA1)	No DNS servers bound
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 and port for the name “sdsu.edu”.  */
status =  nx_dns_info_by_name_get(&my_dns, “sdsu.edu”, &ip_address, &host_port,
                                     4000);

/* If status is NX_SUCCESS the IP address and port for “sdsu.edu” can be found
   in the “ip_address” and “host_port” variables respectively.  */
```

See Also

nx_dns_create, nx_dns_delete, nx_dns_host_by_address_get, nx_dns_server_add, nx_dns_server_remove, nx_dns_server_remove

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 IP address to the previously created DNS instance.

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	Successful DNS Server add
NX_DNS_ERROR	(0xA0)	Internal DNS error.
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 who's IP address is 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

nx_dns_create, nx_dns_delete, nx_dns_host_by_address_get,
nx_dns_host_by_name_get, nx_dns_info_by_name_get,
nx_dns_server_remove, nx_dns_server_remove

nx_dns_server_remove

Remove DNS Server IP Address

Prototype

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

Description

This service removes a DNS Server IP address from the previously created DNS instance.

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

```
/* Rempve a DNS Server who’s 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

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_all

nx_dns_server_remove_all

Removes all DNS Servers

Prototype

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

Description

This service removes all DNS Servers from the previously created DNS instance.

Input Parameters

dns_ptr	Pointer to DNS control block.

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 all DNS Servers from the specified DNS server.  */
status =  nx_dns_server_remove_all(&my_dns);

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

See Also

nx_dns_create, nx_dns_delete, nx_dns_host_by_address_get, nx_dns_host_by_name_get, nx_dns_info_by_name_get, nx_dns_server_add, nx_dns_server_remove