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

Dynamic Host Configuration Protocol for Clients

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1050
Revision 5.0

Contents

Chapter 1 Introduction to DHCP Client 4
Dynamic IP Address Assignment 4
RARP Alternatives 4
DHCP Messages 5
DHCP Communication 5
DHCP Client State Machine 5
DHCP User Request 6
DHCP RFCs 6
Chapter 2 Installation and Use of DHCP Client 7
Product Distribution 7
DHCP Installation 7
Using DHCP 7
In the Bound State 8
Sending DHCP Messages To The Server 8
Starting and Stopping the DHCP Client 9
Using the DHCP Client with Auto IP 9
Packet Chaining 10
Small Example System 11
Multi-Server Environments 13
BOOTP Protocol 13
DHCP Multihome Support 13
Configuration Options 15
Chapter 3 Description of DHCP Client Services 19
nx_dhcp_create 21
nx_dhcp_delete 22
nx_dhcp_force_renew 23
nx_dhcp_request_client_ip 24
nx_dhcp_reinitialize 25
nx_dhcp_release 26
nx_dhcp_send_request 27
nx_dhcp_release, nx_dhcp_state_change_notify, nx_dhcp_reinitialize, nx_dhcp_stop, nx_dhcp_startnx_dhcp_set_interface_index 27
nx_dhcp_start 29
nx_dhcp_state_change_notify 30
nx_dhcp_stop 32
nx_dhcp_user_option_retrieve 33
nx_dhcp_user_option_convert 35

Chapter 1

Introduction to DHCP Client

In NetX, the application's IP address is one of the supplied parameters to the *nx_ip_create* service call. Supplying the IP address poses no problem if the IP address is known to the application, either statically or through user configuration. However, there are some instances where the application doesn't know or care what its IP address is. In such situations, a zero IP address should be supplied to the *nx_ip_create* function and the DHCP Client protocol should be used to dynamically obtain an IP address.

Dynamic IP Address Assignment

The basic service used to obtain a dynamic IP address from the network is the Reverse Address Resolution Protocol (RARP). This protocol is similar to ARP, except it is designed to obtain an IP address for itself instead of finding the MAC address for another network node. The low-level RARP message is broadcast on the local network and it is the responsibility of a server on the network to respond with an RARP response, which contains a dynamically allocated IP address.

Although RARP provides a service for dynamic allocation of IP addresses, it has several shortcomings. The most glaring deficiency is that RARP only provides dynamic allocation of the IP address. In most situations, more information is necessary in order for a device to properly participate on a network. In addition to an IP address, most devices need the network mask and the gateway IP address. The IP address of a DNS server and other network information may also be needed. RARP does not have the ability to provide this information.

RARP Alternatives

In order to overcome the deficiencies of RARP, researchers developed a more comprehensive IP address allocation mechanism called the BOOTstrap Protocol (BOOTP). This protocol has the ability to dynamically allocate an IP address and also provide additional important network information. However, BOOTP has the drawback of being designed for static network configurations. It does not allow for quick or automated address assignment.

This is where the Dynamic Host Configuration Protocol (DHCP) is extremely useful. DHCP is designed to extend the basic functionality of

BOOTP to include completely automated IP server allocation and completely dynamic IP address allocation through “leasing” an IP address to a client for a specified period of time. DHCP can also be configured to allocate IP addresses in a static manner like BOOTP.

DHCP Messages

Although DHCP greatly enhances the functionality of BOOTP, DHCP uses the same message format as BOOTP and supports the same vendor options as BOOTP. In order to perform its function, DHCP introduces seven new DHCP-specific options, as follows:

DISCOVER	(1)	(sent by DHCP Client)
OFFER	(2)	(sent by DHCP Server)
REQUEST	(3)	(sent by DHCP Client)
DECLINE	(4)	(sent by DHCP Server)
ACK	(5)	(sent by DHCP Server)
NACK	(6)	(sent by DHCP Server)
RELEASE	(7)	(sent by DHCP Client)
INFORM	(8)	(sent by DHCP Client)
FORCERENEW	(9)	(sent by DHCP Server)

DHCP Communication

DHCP utilizes the UDP protocol to send requests and field responses. Prior to having an IP address, UDP messages carrying the DHCP information are sent and received by utilizing the IP broadcast address of 255.255.255.255.

DHCP Client State Machine

The DHCP Client is implemented as a state machine. The state machine is processed by an internal DHCP thread that is created during *nx_dhcp_create* processing. The main states of DHCP Client are as follows:

State	Meaning
NX_DHCP_STATE_BOOT	Starting with a previous IP address
NX_DHCP_STATE_INIT	Starting with no previous IP address value
NX_DHCP_STATE_SELECTING	Waiting for a response from any DHCP server

NX_DHCP_STATE_REQUESTING	DHCP Server identified, IP address request sent
NX_DHCP_STATE_BOUND	DHCP IP Address lease established
NX_DHCP_STATE_RENEWING	DHCP IP Address lease renewal time elapsed, renewal requested
NX_DHCP_STATE_REBINDING	DHCP IP Address lease rebind time elapsed, renewal requested
NX_DHCP_STATE_FORCERENEW	DHCP IP Address lease established, force renewal by server or by application

DHCP User Request

Once the DHCP server grants an IP address, the DHCP client processing can request additional parameters — one at a time — by using the *nx_dhcp_user_option_request* service.

DHCP RFCs

NetX DHCP is compliant with RFC2132, RFC2131, and related RFCs.

Chapter 2

Installation and Use of DHCP Client

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

Product Distribution

DHCP 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_dhcp.h	Header file for DHCP for NetX
nx_dhcp.c	C Source file for DHCP for NetX
nx_dhcp.pdf	PDF description of DHCP for NetX
demo_netx_dhcp.c	NetX DHCP demonstration

DHCP Installation

In order to use DHCP 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_dhcp.h* and *nx_dhpc.c* files should be copied into this directory.

Using DHCP

Using DHCP for NetX is easy. Basically, the application code must include *nx_dhcp.h* after it includes *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX, respectively. Once *nx_dhcp.h* is included, the application code is then able to make the DHCP function calls specified later in this guide. The application must also include *nx_dhcp.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 DHCP.

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

To obtain a previously assigned IP address, the DHCP Client can initiate the DHCP process with the Request message and Option 50 "Requested IP Address" to the DHCP Server. The DHCP Server will respond with

either an ACK message if it grants the IP address to the Client or a NACK if it refuses. In the latter case, the DHCP Client restarts the DHCP process at the Init state with a Discover message and no requested IP address. The host application first creates the DHCP Client, then calls the *nx_dhcp_request_client_ip* API service to set the requested IP address before starting the DHCP process with *nx_dhcp_start*. An example DHCP application is provided elsewhere in this document for more details.

In the Bound State

While the DHCP client is in the bound state, the client thread as it iterates once per interval (as specified by NX_DHCP_TIME_INTERVAL) will decrement the time out on the IP lease assigned to the Client. When the renewal time as elapsed it will forward the DHCP client into the RENEW state to request a renewal from the DHCP server. There is an option to periodically check for DHCP Server messages. This is set by the NX_DHCP_TIMEOUT_DECREMENTS option which determines the intervals between checking for messages as follows:

```
NX_DHCP_TIMEOUT_DECREMENTS * NX_DHCP_TIME_INTERVAL
```

When this amount of time has elapsed since either the IP lease was issued or since the last check for Server messages, the DHCP Client will check the receive queue for DHCP messages.

The default setting for `NX_DHCP_TIMEOUT_DECREMENTS` is 0xFFFFFFFF which indicates do not check for Server messages.

Sending DHCP Messages To The Server

The DHCP Client has API services that allow the host application to send a message to the DHCP Server. Note these services are NOT intended for the host application to manually run the DHCP Client protocol as they primarily send the message without necessarily updating the DHCP Client internal state.

- *nx_dhcp_release*: this sends a release message to the Server when the host application is either leaving the network or needs relinquish its IP address.

- *nx_dhcp_forcerenew*: this does not send a message but sets the DHCP Client in the FORCERENEW state if the Server sends the Client a FORCERENEW message. The DHCP Client will then set itself to the RENEW state to begin requesting IP lease renewal.

- *nx_dhcp_send_request*: This takes as an argument a DHCP message type, as specified in nx_dhcp.h, and sends the message to the Server. This is how a host application would send a DECLINE or INFORM_REQUEST to the Server.

See "*Description of DHCP Services*" for more information about these services elsewhere in this document.

Starting and Stopping the DHCP Client

To stop the DHCP Client, regardless if it has achieved a bound state, the host application calls *nx_dhcp_stop*. This will wait for the DHCP Client to pause between its loop iterations and give other threads, e.g. the host application, a chance to access the DHCP Client profile (DHCP state, IP address, etc) and even send messages back to the Server.

To restart a DHCP client, the host application must first stop the DHCP Client using the *nx_dhcp_stop* service described above. Then the host can call nx_dhcp_start to resume the DHCP Client. If the host application wishes to clear a previous DHCP Client profile, for example, one obtained from a previous DHCP Server on another network, the host application should call *nx_dhcp_reinitialize* to perform this task internally before calling nx_dhcp_start.

A typical sequence might be:

```
nx_dhcp_stop(&my_dhcp);

nx_dhcp_reinitialize(&my_dhcp);

nx_dhcp_start(&my_dhcp);
```

Note that while the DHCP Client is stopped, the timer on the IP lease expiration is stopped as well, so stopping the DHCP Client is not advised unless the host application requires rebooting or switching networks.

Using the DHCP Client with Auto IP

The NetX DHCP Client works concurrently with the Auto IP protocol in applications where DHCP and Auto IP guarantee an address where a DHCP Server is not guaranteed to be available or responding. However, If the host is unable to detect a Server or get an IP address assigned, it can switch to the Auto IP protocol for a local IP address. However before

doing so, it is advisable to stop the DHCP Client temporarily while Auto IP goes through the “probe” and “defense” stages. Once an Auto IP address is assigned to the host, the DHCP Client can be restarted and if a DHCP Server does become available, the host IP address can accept the IP address offered by the DHCP Server while the application is running.

The NetX Auto IP has an address change notification for the host to monitor its activities in the event of an IP address change.

Packet Chaining

For more efficient use of packet pool and memory resources, the DHCP Client can handle incoming chained packets (datagrams exceeding the driver MTU) from the Ethernet driver. If the driver has this capability, the host application can set the packet pool for receiving packets to below the mandatory 592 bytes of DHCP message data the DHCP Client is expected to handle from the DHCP Server, as per RFC 2131 which includes 548 bytes of DHCP data and IP, UDP and Ethernet headers.

Note that the host application can optimize the packet payload and number of packets in the packet pool that is part of the DHCP Client, and which is used for sending DHCP messages out. It can optimize the size based on expected usage and size of the DHCP Client messages.

Small Example System

An example of how easy it is to use NetX is described in Figure 1.1 that appears below. In this example, the DHCP include file *nx_dhcp.h* is brought in at line 3. Next, DHCP is created "*my_thread_entry*" at line 101. Note that the DHCP control block "*my_dhcp*" was defined as a global variable at line 9 previously. After successful creation, the DHCP process of requesting an IP address is initiated at the call to *nx_dhcp_start* at line 108. It is here that attempts are initiated to contact the DHCP server. At this point, the application code waits for a valid IP address to appear using the *nx_ip_status_check* service starting at line 95. After line 127, DHCP has received a valid IP address and the application can then proceed, utilizing NetX TCP/IP services as desired.

```
0001 #include   "tx_api.h"
0002 #include   "nx_api.h"
0003 #include   "nx_dhcp.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_DHCP                 my_dhcp;
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 without an IP address. */
0057     status = nx_ip_create(&my_ip, "My NetX IP Instance", IP_ADDRESS(0,0,0,0),
```

```
0058            0xFFFFFF00, &my_pool, my_netx_driver, pointer,
0059            DEMO_STACK_SIZE, 1);
0060    pointer =  pointer + DEMO_STACK_SIZE;
0061
0062    /* Check for IP create errors.  */
0063    if (status)
0064        error_counter++;
0065
0066    /* Enable ARP and supply ARP cache memory for my IP Instance.  */
0067    status =  nx_arp_enable(&my_ip, (void *) pointer, 1024);
0068    pointer = pointer + 1024;
0069
0070    /* Check for ARP enable errors.  */
0071    if (status)
0072        error_counter++;
0073
0074    /* Enable UDP.  */
0075    status =  nx_udp_enable(&my_ip);
0076    if (status)
0077        error_counter++;
0078 }
0079
0080
0081 /* Define my thread.  */
0082
0083 void    my_thread_entry(ULONG thread_input)
0084 {
0085
0086 UINT        status;
0087 ULONG       actual_status;
0088 NX_PACKET   *my_packet;
0089
0090    /* Wait for the link to come up.  */
0091    do
0092    {
0093
0094        /* Get the link status.  */
0095        status =  nx_ip_status_check(&my_ip, NX_IP_LINK_ENABLED,
0096                                                &actual_status, 100);
0097
0098    } while (status != NX_SUCCESS);
0099
0100    /* Create a DHCP instance.  */
0101    status =  nx_dhcp_create(&my_dhcp, &my_ip, "My DHCP");
0102
0103    /* Check for DHCP create error.  */
0104    if (status)
0105        error_counter++;
0106
0107    /* Start DHCP.  */
0108    nx_dhcp_start(&my_dhcp);
0109
0110    /* Check for DHCP start error.  */
0111    if (status)
0112        error_counter++;
0113
0114    /* Wait for IP address to be resolved through DHCP.  */
0115    nx_ip_status_check(&my_ip, NX_IP_ADDRESS_RESOLVED,
0116                                        (ULONG *) &status, 100000);
0117
0118    /* Check to see if we have a valid IP address.  */
0119    if (status)
0120    {
0121        error_counter++;
0122        return;
0123    }
0124    else
0125    {
0126
0127          /* Yes, a valid IP address is now on lease…  All NetX
0128             services are available.
0129    }
0130 }
```

Figure 1.1 Example of DHCP use with NetX

Multi-Server Environments

On networks where there is more than one DHCP Server, the DHCP Client accepts the first received DHCP Server Offer message, advances to the Request state, and ignores any other received offers.

The DHCP Client can be configured to send an ARP probe after IP address assignment to verify the IP address is unique. This is recommended by RFC 2131 and is particularly important in environments with more than one DHCP Server. If the host application enables the NX_DHCP_CLIENT_SEND_ARP_PROBE option (and optionally adjusts the NX_DHCP_ARP_ PROBE_TIMEOUT), the DHCP Client will send a 'self addressed' ARP probe and wait for the specified time for a response. If none is received, the DHCP Client advances to the Bound state. If a response is received, the DHCP Client assumes the address is already in use. It automatically sends a DECLINE message to the Server, and returns to the Client to the INIT state. This restarts the DHCP state machine and the Client sends another DISCOVER message to the Server.

BOOTP Protocol

The DHCP Client also supports the BOOTP protocol as well the DHCP protocol. To enable this option and use BOOTP instead of DHCP, the host application must set the NX_DHCP_BOOTP_ENABLE configuration option. The host application can still request specific IP addresses in the BOOTP protocol. However, the DHCP Client does not support loading the host operating system as BOOTP is sometimes used to do.

DHCP Multihome Support

DHCP Client v5.1 and later supports multihomed devices. Multihome support is available starting in NetX 5.3 and NetX Duo 5.6. For singly homed devices, DHCP for NetX defaults to the IP task primary interface, so is backward compatible with previous versions of NetX. Existing host applications will require no changes to work with DHCP Client v5.1.

To run a DHCP Client on a secondary network interface, the host application must set the interface index of the DHCP Client to the secondary interface using the *nx_dhcp_set_interface_index* API service. The interface must already be attached to the primary network interface using the *nx_ip_interface_attach* NetX API call. See the NetX User Guide for more details on multihome support.

If a host requires DHCP to run on both interfaces, it should create a DHCP Client task for each interface, but requires only one IP task interface. Below in Figure 1.2 is an example system on which the host application connects to the DHCP server on its secondary interface. On line 68, the secondary interface is attached to the IP task with a null IP address. On line 104, after the DHCP Client instance is created, the DHCP Client interface index is set to 1 (e.g. the offset from the primary interface which itself is index 0) by calling *nx_dhcp_set_interface_index*. Then the DHCP Client is ready to be started in line 108.

```
0001 #include   "tx_api.h"
0002 #include   "nx_api.h"
0003 #include   "nx_dhcp.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_DHCP                 my_dhcp;
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 without an IP address. */
0057     status = nx_ip_create(&my_ip, "My NetX IP Instance", IP_ADDRESS(0,0,0,0),
0058                0xFFFFFF00, &my_pool, my_netx_driver, pointer, STACK_SIZE, 1);
0059     pointer =  pointer + DEMO_STACK_SIZE;
0060
0061     /* Check for IP create errors.  */
0062     if (status)
```

```
0063         error_counter++;
0064
0065     status =  _nx_ip_interface_attach(&ip_0, "port_2", IP_ADDRESS(0, 0, 0, 0),
                            0xFFFFFF00UL, my_netx_driver);

0066     /* Enable ARP and supply ARP cache memory for my IP Instance.  */
0067     status =  nx_arp_enable(&my_ip, (void *) pointer, 1024);
0068     pointer = pointer + 1024;
0069
0070     /* Check for ARP enable errors.  */
0071     if (status)
0072         error_counter++;
0073
0074     /* Enable UDP.  */
0075     status =  nx_udp_enable(&my_ip);
0076     if (status)
0077         error_counter++;
0078 }
0079
0080
0081 void    my_thread_entry(ULONG thread_input)
0082 {
0083
0084 UINT        status;
0085 ULONG       status;
0086 NX_PACKET   *my_packet;
0087
0088     /* Wait for the link to come up.  */
0089     do
0090     {
0091
0092         /* Get the link status.  */
0093         status =  nx_ip_status_check(&my_ip,NX_IP_LINK_ENABLED,& status,100);
0094     } while (status != NX_SUCCESS);
0095
0096     /* Create a DHCP instance.  */
0097     status =  nx_dhcp_create(&my_dhcp, &my_ip, "My DHCP");
0098
0099     /* Check for DHCP create error.  */
0100     if (status)
0101         error_counter++;
0102
0103     /* Set the DHCP client interface to the secondary interface.
0104     status = nx_dhcp_set_interface_index(&my_dhcp, 1);
0105
0106
0107     /* Start DHCP.  */
0108     nx_dhcp_start(&my_dhcp);
0109
0110     /* Check for DHCP start error.  */
0111     if (status)
0112         error_counter++;
0113
0114     /* Wait for IP address to be resolved through DHCP.  */
0115     nx_ip_status_check(&my_ip, NX_IP_ADDRESS_RESOLVED,
0116                                         (ULONG *) &status, 100000);
0117
0118     /* Check to see if we have a valid IP address.  */
0119     if (status)
0120     {
0121         error_counter++;
0122         return;
0123     }
0124     else
0125     {
0126
0127           /* Yes, a valid IP address is now on lease…  All NetX
0128              services are available.
0129     }
0130 }
```

Figure 1.2 Example of DHCP for NetX with multihome support

Configuration Options

User configurable DHCP options in *nx_dhcp.h* allow the host application to fine tune DHCP Client for its particular requirements. The following is a list of these parameters:

Define	Meaning
NX_PACKET_ALLOCATE_TIMEOUT	Specifies the time out option for allocating a packet from the DHCP Client packet pool. The default value is one second.
NX_DHCP_ENABLE_BOOTP	Defined, this option enables the BOOTP protocol instead of DHCP. By default this option is disabled.
NX_DHCP_ARP_PROBE_TIMEOUT	Specifies the time out option in timer tick to wait for response to the DHCP Client ARP probe (see NX_DHCP_CLIENT_SEND_ARP_PROBE option). If NX_DHCP_CLIENT_SEND_ARP_PROBE is not enabled, this option has no meaning. The value is defaulted to 1000 ticks.
NX_DHCP_CLIENT_SEND_ARP_PROBE	Defined, this enables the DHCP Client to send an ARP probe after IP address assignment to verify the assigned DHCP address is not owned by another host. By default, this option is disabled.
NX_DHCP_FRAGMENT_OPTION	Fragment enable for DHCP UDP requests. By default, this value is NX_DONT_FRAGMENT to disable DNS UDP fragmenting.
NX_DHCP_MAX_RETRANS_TIMEOUT	Specifies the maximum wait option for receiving a DHCP Server reply to client message before retransmitting the message. The default value is the RFC 2131 recommended 64 seconds.

NX_DHCP_MIN_RENEW_TIMEOUT	Specifies minimum wait option for receiving a DHCP Server message and sending a renewal request after the DHCP Client is bound to an IP address. The default value is 60 seconds. However, the DHCP Client uses the renew and rebind expiration times from the DHCP server message before defaulting to the minimum renew timeout.
NX_DHCP_MIN_RETRANS_TIMEOUT	Specifies the minimum wait option for receiving a DHCP Server reply to client message before retransmitting the message. The default value is the RFC 2131 recommended 4 seconds.
NX_DHCP_PACKET_PAYLOAD	Specifies the size in bytes of the DHCP Client packet. The default value and maximum allowed by DHCP protocol (RFC 2131) is 592 bytes, (548 bytes DHCP message + IP, UDP and Ethernet headers).
NX_DHCP_PACKET_POOL_SIZE	Specifies the size of the DHCP Client packet pool. The default value is (5 *NX_DHCP_PACKET_PAYLOAD) which will provide four packets plus room for internal packet pool overhead.
NX_DHCP_THREAD_PRIORITY	Priority of the DHCP thread. By default, this value specifies that the DHCP thread runs at priority 1.
NX_DHCP_THREAD_STACK_SIZE	Size of the DHCP thread's stack. By default, the size is 1024, which represents a stack of 1024 bytes.

NX_DHCP_TIMEOUT_DECREMENTS Determines how long the DHCP client waits between checking for DHCP server messages once the Client has reached the bound state as follows. The interval is defined as:

NX_DHCP_TIMEOUT_DECREMENTS * NX_DHCP_TIME_INTERVAL

The default value is 0xFFFFFFFF for disabled. The Client periodically decrements the time remaining on the lease and if expired moves the Client into the RENEW/ REQUEST state.

NX_DHCP_TIME_INTERVAL Number of seconds between iterations of the DHCP client entry thread function. By default, this value is 1 second updates.

NX_DHCP_TIME_TO_LIVE Specifies the number of routers this packet can pass before it is discarded. The default value is set to 0x80.

NX_DHCP_TYPE_OF_SERVICE Type of service required for the DHCP UDP requests. By default, this value is defined as NX_IP_NORMAL to indicate normal IP packet service.

Chapter 3

Description of DHCP Client Services

This chapter contains a description of all NetX DHCP 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_dhcp_create
Create a DHCP instance

nx_dhcp_delete
Delete a DHCP instance

nx_dhcp_force_renew
Handle Server force renew message

nx_dhcp_release
Send *Release message to server*

nx_dhcp_reinitialize
Clear DHCP client network parameters

nx_dhcp_request_client_ip
Specify a specific IP address

nx_dhcp_send_request
Send DHCP message to server

nx_dhcp_set_interface_index
Specify the Client network interface

nx_dhcp_start
Start DHCP processing

nx_dhcp_state_change_notify
Notify application of DHCP state change

nx_dhcp_stop
Stop DHCP processing

nx_dhcp_user_option_retrieve
Retrieve DHCP option

nx_dhcp_user_option_convert
Convert four bytes to ULONG

nx_dhcp_create

Create a DHCP instance

Prototype

```
UINT nx_dhcp_create(NX_DHCP *dhcp_ptr, NX_IP *ip_ptr, CHAR *name_ptr);
```

Description

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

Important Note: The application must make sure it is capable of handling a 576 byte UDP message including the UDP, IP and Ethernet headers.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
ip_ptr	Pointer to previously created IP instance.
name_ptr	Pointer to name for DHCP instance.

Return Values

status		Status return from NetX
NX_SUCCESS	(0x00)	Successful DHCP create
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_NOT_ENABLED	(0x14)	UDP not enabled on IP instance

Allowed From

Threads

Example

```
/* Create a DHCP instance.  */
status =  nx_dhcp_create(&my_dhcp, &my_ip, "My DHCP");

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

See Also

nx_dhcp_delete, nx_dhcp_request_client_ip, nx_dhcp_set_interface_index, nx_dhcp_release, nx_dhcp_start, nx_dhcp_state_change_notify, nx_dhcp_stop

nx_dhcp_delete

Delete a DHCP instance

Prototype

```
UINT nx_dhcp_delete(NX_DHCP *dhcp_ptr);
```

Description

This service deletes a previously created DHCP instance.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP delete.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Delete a DHCP instance.  */
status =  nx_dhcp_delete(&my_dhcp);

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

See Also

nx_dhcp_create, nx_dhcp_release, nx_dhcp_start,
nx_dhcp_state_change_notify, nx_dhcp_stop

nx_dhcp_force_renew

Handle a server force renew message

Prototype

```
UINT nx_dhcp_force_renew(NX_DHCP *dhcp_ptr);
```

Description

This service enables the host application to handle a force renew message. It sets the DHCP client to the FORCERENEW state so that on the next DHCP client thread iteration it will execute the Client in the RENEW state and obtain a new IP lease.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP release.
NX_DHCP_NOT_BOUND	(0x94)	The IP address has not been leased so it can't be released.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Handle a force renew message from server.  */
status =  nx_dhcp_force_renew(&my_dhcp);

/* If status is NX_SUCCESS the DHCP client state is the FORCE RENEW state.  */
```

See Also

nx_dhcp_create, nx_dhcp_delete, nx_dhcp_start, nx_dhcp_state_change_notify, nx_dhcp_stop

nx_dhcp_request_client_ip

Set requested IP address for DHCP instance

Prototype

```
UINT nx_dhcp_request_client_ip(NX_DHCP *dhcp_ptr,
                  ULONG client_ip_address, UINT skip_discover_message);
```

Description

This service sets the IP address for the DHCP instance to request from the DHCP Server. If the *skip_discover_message* flag is set, the DHCP client skips the discover message and sends a Request message.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
client_ip_address	IP address to request from DHCP server
skip_discover_message	If true, DHCP Client sends Request message; else it starts with the Discover message.

Return Values

NX_SUCCESS	(0x00)	Requested IP address is set.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer

Allowed From

Threads

Example

```
/* Set the DHCP Client requested IP address and skip the discover message.  */
status =  nx_dhcp_request_client_ip(&my_dhcp, IP(192,168,0,6), NX_TRUE);
/* If status is NX_SUCCESS requested IP address was successfully set.  */
```

See Also

nx_dhcp_delete, nx_dhcp_create, nx_dhcp_release, nx_dhcp_start, nx_dhcp_state_change_notify, nx_dhcp_stop

nx_dhcp_reinitialize

Clear the DHCP client network parameters

Prototype

```
UINT nx_dhcp_reinitialize(NX_DHCP *dhcp_ptr);
```

Description

This service clears the host application network parameters (IP address, network address and network mask), and returns the DHCP client to the INIT state. It is used in combination with *nx_dhcp_stop* and *nx_dhcp_start* to 'restart' a host on another network with another server:

```
nx_dhcp_stop(&my_dhcp);
nx_dhcp_reinitialize(&my_dhcp);
nx_dhcp_start(&my_dhcp);
```

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP release
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer

Allowed From

Threads

Example

```
/* Reinitialize the previously started DHCP client.  */
status =  nx_dhcp_reinitialize(&my_dhcp);

/* If status is NX_SUCCESS the host application successfully reinitialized its
network parameters and DHCP client state. */
```

See Also

nx_dhcp_create, nx_dhcp_delete, nx_dhcp_start, nx_dhcp_state_change_notify, nx_dhcp_stop

nx_dhcp_release

Release Leased IP address

Prototype

```
UINT nx_dhcp_release(NX_DHCP *dhcp_ptr);
```

Description

This service releases the IP address obtained from the previous DHCP start request and returns the DHCP state machine to the initial state. A new IP address can be requested by calling *nx_dhcp_start* again.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP release.
NX_DHCP_NOT_BOUND	(0x94)	The IP address has not been leased so it can't be released.
NX_DHCP_NOT_STARTED	(0x96)	The DHCP instance not started.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Release the previously leased IP address.  */
status =  nx_dhcp_release(&my_dhcp);

/* If status is NX_SUCCESS the previous IP lease was successfully released.  */
```

See Also

nx_dhcp_create, nx_dhcp_delete, nx_dhcp_start, nx_dhcp_state_change_notify, nx_dhcp_stop

nx_dhcp_send_request

Send DHCP message to Server

Prototype

```
UINT nx_dhcp_send_request(NX_DHCP *dhcp_ptr, UINT dhcp_message_type);
```

Description

This service sends a message to the DHCP server. This is intended for the host application to send a DECLINE message, ask (INFORM_REQUEST) for network information, or other messages. It is NOT intended for the host application to drive the DHCP Client state machine. It does not update the DHCP client internally.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
dhcp_message_type	Message request (defined in *nx_dhcp.h*)

Return Values

status	variable	Actual completion status
NX_DHCP_NOT_STARTED	(0x96)	Invalid interface index
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Send the DECLINE message back to the server; this should be done if the DHCP
Client discovers the assigned IP address is already owned.  */
status =  nx_dhcp_send_request(&my_dhcp, NX_DHCP_TYPE_DHCPDECLINE);
/* If status is NX_SUCCESS a DHCP message was successfully sent.  */
```

See Also

nx_dhcp_release, nx_dhcp_state_change_notify, nx_dhcp_reinitialize, nx_dhcp_stop, nx_dhcp_start

nx_dhcp_set_interface_index

Set network interface for DHCP instance

Prototype

```
UINT nx_dhcp_set_interface_index(NX_DHCP *dhcp_ptr, UINT index);
```

Description

This service sets the network interface DHCP instance connects to the DHCP Server on.

Important Note: The application must previously attach the specified interface to the IP task.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
index	Index of device network interface

Return Values

NX_SUCCESS	(0x00)	Interface is successfully set.
NX_DHCP_BAD_INTERFACE_INDEX _ERROR	(0x9A)	Invalid interface index
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer

Allowed From

Threads

Example

```
/* Set the DHCP Client interface to the secondary interface (index 1).  */
status =  nx_dhcp_set_interface_index(&my_dhcp, 1);
/* If status is NX_SUCCESS a DHCP interface was successfully set.  */
```

See Also

nx_dhcp_delete, nx_dhcp_request_client_ip, nx_dhcp_create, nx_dhcp_release, nx_dhcp_start, nx_dhcp_state_change_notify, nx_dhcp_stop

nx_dhcp_start

Start DHCP processing

Prototype

```
UINT nx_dhcp_start(NX_DHCP *dhcp_ptr);
```

Description

This service starts DHCP processing, which includes contacting the DHCP server on the network in order to obtain an IP address.

Note that when proceeding further, the application should use *nx_ip_status_check* to see when an IP address is obtained.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP start.
NX_DHCP_ALREADY_STARTED	(0x93)	The DHCP instance has already been started.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of service.

Allowed From

Threads

Example

```
/* Start the DHCP processing for this IP instance.  */
status =  nx_dhcp_start(&my_dhcp);

/* If status is NX_SUCCESS the DHCP was successfully started.  */
```

See Also

nx_dhcp_create, nx_dhcp_delete, nx_dhcp_release, nx_dhcp_state_change_notify, nx_dhcp_stop, nx_dhcp_request_client_ip, nx_dhcp_set_interface_index

nx_dhcp_state_change_notify

Notify application of DHCP state change

Prototype

```
UINT nx_dhcp_state_change_notify(NX_DHCP *dhcp_ptr,
  VOID (*dhcp_state_change_notify)(NX_DHCP *dhcp_ptr,UCHAR new_state));
```

Description

This service registers the specified application callback function with DHCP. Once this service is called, the specified callback function is invoked whenever the DHCP state changes. Following are values associated with the various DHCP states:

State	Value
NX_DHCP_STATE_BOOT	1
NX_DHCP_STATE_INIT	2
NX_DHCP_STATE_SELECTING	3
NX_DHCP_STATE_REQUESTING	4
NX_DHCP_STATE_BOUND	5
NX_DHCP_STATE_RENEWING	6
NX_DHCP_STATE_REBINDING	7
NX_DHCP_STATE_FORCERENEW	8

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.
dhcp_state_change_notify	Application callback function pointer

Return Values

NX_SUCCESS	(0x00)	Successful DHCP start.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of service.

Allowed From

Threads

Example

```
/* Register the "my_state_change" function to be called on any DHCP state change,
assuming DHCP has already been created.  */
status =  nx_dhcp_state_change_notify(&my_dhcp, my_state_change);
```

```
/* If status is NX_SUCCESS the callback function was successfully
   registered.  */
```

See Also

nx_dhcp_create, nx_dhcp_start, nx_dhcp_stop, nx_dhcp_user_option_retrieve, nx_dhcp_user_option_convert

nx_dhcp_stop

Stops DHCP processing

Prototype

```
UINT nx_dhcp_stop(NX_DHCP *dhcp_ptr);
```

Description

This service stops DHCP processing, which includes sending a release request to the DHCP server on the network if DHCP is in a bound state.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP stop
NX_DHCP_NOT_STARTED	(0x96)	The DHCP instance not started.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of service.

Allowed From

Threads

Example

```
/* Stop the DHCP processing for this IP instance.  */
status =  nx_dhcp_stop(&my_dhcp);

/* If status is NX_SUCCESS the DHCP was successfully stopped.  */
```

See Also

nx_dhcp_create, nx_dhcp_delete, nx_dhcp_release, nx_dhcp_start, nx_dhcp_state_change_notify

nx_dhcp_user_option_retrieve

Retrieve a DHCP option from last server response

Prototype

```
UINT nx_dhcp_user_option_retrieve(NX_DHCP *dhcp_ptr,
                  UINT request_option, UCHAR *destination_ptr,
                                       UINT *destination_size);
```

Description

This service retrieves the specified DHCP option from the server's last message. If successful, the option response string returned is copied into the specified application buffer.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.
request_option	DHCP option, as specified by the RFCs. See the **NX_DHCP_OPTION*** defines in *nx_dhcp.h*.
destination_ptr	Pointer to the destination for the response string.
destination_size	Pointer to the size of the destination and on return, the destination to place the number of bytes returned.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP option retrieval.
NX_DHCP_NOT_BOUND	(0x94)	The IP address has not been leased yet so option requests cannot be made.
NX_DHCP_ERROR	(0x90)	Option not found in buffer. Please include the option in the **_nx_dhcp_request_parameters** which is defined at the top of **nx_dhcp.c**.
NX_DHCP_DEST_TO_SMALL	(0x95)	Destination is too small to hold response.
NX_PTR_ERROR	(0x16)	Invalid DHCP or destination pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
UCHAR  dns_ip_string[4];
ULONG  size;

/* Obtain the IP address of the DNS server.  */
size =    sizeof(dnx_ip_string);
status =  nx_dhcp_user_option_retrieve(&my_dhcp, NX_DHCP_OPTION_DNS_SVR,
              dns_ip_string, &size);

/* If status is NX_SUCCESS the DNS IP address is in dns_ip_string.  */
```

See Also

nx_dhcp_user_option_convert

nx_dhcp_user_option_convert

Convert four bytes to ULONG

Prototype

```
ULONG nx_dhcp_user_option_convert(UCHAR *option_string_ptr);
```

Description

This service converts the four characters pointed to by “option_string_ptr” into an unsigned long value. It is especially useful when IP addresses are present.

Input Parameters

option_string_ptr	Pointer to previously retrieved option string.

Return Values

Value	Value of first four bytes.

Allowed From

Threads

Example

```
UCHAR  dns_ip_string[4];
ULONG  dns_ip;

/* Convert the first four bytes of “dns_ip_string” to an actual IP
   address in “dns_ip.”  */
dns_ip =  nx_dhcp_user_option_convert(dns_ip_string);

/* If status is NX_SUCCESS the DNS IP address is in “dns_ip.”  */
```

See Also

nx_dhcp_stop, nx_dhcp_user_option_retrieve