NETX™

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 5.12

Contents

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 Client)
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 Client)

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 (currently not supported) or by the application calling nx_dhcp_force_renew
NX_DHCP_STATE_ADDRESS_PROBING	DHCP IP Address probing, send the ARP probe to detect IP address conflict.

DHCP Client Multiple Interface Support

The DHCP Client was previously implemented to run on only a single network interface. The default behavior was (and still is) for the DHCP Client to run on the primary interface. By calling *nx_dhcp_set_interface_index*, the application could (and still can) run DHCP on a secondary network interface instead of the primary interface.

It now supports DHCP running on multiple interfaces in parallel. See **DHCP Client on Multiple Interfaces Simultaneously** in Chapter Two for specific details how to run DHCP Client on more than one physical interface simultaneously.

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 Client Socket Queue

The DHCP Client automatically clears broadcast packets from DHCP Servers intended for other DHCP Clients from its socket receive queue while waiting for Server to respond to itself. In a busy network, not doing so could cause packets intended for the Client to be dropped.

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
demo_netx_multihome_dhcp_client.c	NetX DHCP Client demonstration of DHCP on multiple interfaces

DHCP Installation

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_dhcp.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 DHCP Client thread processes the Client state once per interval (as specified by NX_DHCP_TIME_INTERVAL) and decrements the time remaining on the IP lease assigned to the Client. When the renewal time has elapsed the DHCP Client state is updated to the RENEW state where the Client will request a renewal from the DHCP Server.
.

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_decline*: this sends a DECLINE message to the Server if the host application determines independently of the DHCP Client that its IP address is already in use.

- *nx_dhcp_forcerenew*: this sends a FORCERENEW message to the Server

- *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 intended primarily for sending the DHCP INFORM message.

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*.

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);
```

For DHCP applications running on only a single DHCP interface, stopping the DHCP Client also inactivates the DHCP CLIENT timer. Thus it is no longer keeping track of the time remaining on the IP lease. Stopping DHCP Client on a particular interface will not inactivate the DHCP Client timer but will stop timer updates to the time remaining on the IP lease on that interface

Therefore, 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.

Small Example System

An example of how use NetX is described in Figure 1.1 below. The DHCP Client is created "*my_thread_entry*" at line 101. After successful creation, the DHCP process is initiated at the call to *nx_dhcp_start* at line 108. At this point the DHCP Client attempts are initiated to contact the DHCP server. During this process, the application code waits for a valid IP address to be registered with the IP instance using the *nx_ip_status_check* service (or *nx_ip_interface_status_check* for a secondary interface) starting at line 95. This is more commonly done in a loop with a shorter wait option.

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 intmain()
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,
                  pointer, DEMO_STACK_SIZE,
                  2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
    pointer =  pointer + DEMO_STACK_SIZE;

    /* Initialize the NetX system.  */
    nx_system_initialize();

    /* Create a packet pool.  */
    status =  nx_packet_pool_create(&my_pool, "NetX Main Packet Pool",
                                          1024, pointer, 64000);
    pointer = pointer + 64000;

    /* Check for pool creation error.  */
    if (status)
        error_counter++;

    /* Create an IP instance without an IP address. */
    status = nx_ip_create(&my_ip, "My NetX IP Instance", IP_ADDRESS(0,0,0,0),
            0xFFFFFF00, &my_pool, my_netx_driver, pointer,
            DEMO_STACK_SIZE, 1);
    pointer =  pointer + DEMO_STACK_SIZE;

    /* 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       actual_status;
NX_PACKET   *my_packet;

    /* Wait for the link to come up.  */
    do
    {

        /* Get the link status.  */
        status =  nx_ip_status_check(&my_ip, NX_IP_LINK_ENABLED,
                                                &actual_status, 100);

    } while (status != NX_SUCCESS);

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

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

    /* Start DHCP.  */
    nx_dhcp_start(&my_dhcp);

    /* Check for DHCP start error.  */
    if (status)
        error_counter++;

    /* Wait for IP address to be resolved through DHCP.  */
    nx_ip_status_check(&my_ip, NX_IP_ADDRESS_RESOLVED,
                                        (ULONG *) &status, 100000);

    /* Check to see if we have a valid IP address.  */
    if (status)
    {
        error_counter++;
        return;
    }
    else
    {

          /* Yes, a valid IP address is now on lease…  All NetX
             services are available.
    }
}
```

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.

ARP Probes

The DHCP Client can be configured to send one or more ARP probes after IP address assignment from the DHCP Server to verify the IP address is not already in use. The ARP probe step 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 (see **Configuration Options** in Chapter Two for additional ARP probe options), 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 reinitializes the Client to restart the DHCP probes again from 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 on a Secondary Interface

The NetX DHCP Client can run on secondary interfaces rather than the default primary interface.

To run NetX 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* service. See the NetX User Guide for more details on attaching secondary interfaces.

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 65, 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 intmain()
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);
```

```
    pointer =  pointer + DEMO_STACK_SIZE;

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

    status =  _nx_ip_interface_attach(&ip_0, "port_2", IP_ADDRESS(0, 0, 0,0),
                         0xFFFFFF00UL, my_netx_driver);

    /* 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++;
}


void    my_thread_entry(ULONG thread_input)
{

UINT        status;
ULONG       status;
NX_PACKET   *my_packet;

    /* Wait for the link to come up.  */
    do
    {

        /* Get the link status.  */
        status =  nx_ip_status_check(&my_ip,NX_IP_LINK_ENABLED,& status,100);
    } while (status != NX_SUCCESS);

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

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

    /* Set the DHCP client interface to the secondary interface.
   status = nx_dhcp_set_interface_index(&my_dhcp, 1);


    /* Start DHCP.  */
    nx_dhcp_start(&my_dhcp);

    /* Check for DHCP start error.  */
    if (status)
        error_counter++;

    /* Wait for IP address to be resolved through DHCP.  */
    nx_ip_status_check(&my_ip, NX_IP_ADDRESS_RESOLVED,
                                        (ULONG *) &status, 100000);

    /* Check to see if we have a valid IP address.  */
    if (status)
    {
        error_counter++;
        return;
    }
    else
    {

          /* Yes, a valid IP address is now on lease…  All NetX
             services are available.
    }
}
```

Figure 1.2 Example of DHCP for NetX with multihome support

DHCP Client on Multiple Interfaces Simultaneously

To run DHCP Client on multiple interfaces, NX_MAX_PHYSICAL_INTERFACES in *nx_api.h* must be set to the number of physical interfaces connected to the device. By default, this value is 1 (e.g. the primary interface). To register an additional interface to the IP instance use the *nx_ip_interface_attach* service. See the NetX User Guide for more details on attaching secondary interfaces.

The next step is to set the NX_DHCP_CLIENT_MAX_RECORDS in *nx_dhcp.h* to the maximum number of interfaces expected to run DHCP simultaneously. Note that NX_DHCP_CLIENT_MAX_RECORDS does not have to equal NX_MAX_PHYSICAL_INTERFACES. For example, NX_MAX_PHYSICAL_INTERFACES can be 3 and NX_DHCP_CLIENT_MAX_RECORDS equal to 2. In this configuration, only two interfaces (and they can be any two of the three physical interfaces at any time) of the three physical interfaces can run DHCP at any one time. DHCP Client Records do not have a one to one mapping to network interfaces e.g. Client Record 1 does not automatically correlate to physical interface index 1.

NX_DHCP_CLIENT_MAX_RECORDS can also be set to greater than NX_MAX_PHYSICAL_INTERFACES but this would create unused client records and be an inefficient use of memory.

Before it can start DHCP on any interface, the application must enable those interfaces by calling *nx_dhcp_interface_enable*. Note that the exception is the primary interface which is automatically enabled in the *nx_dhcp_create* call (and which can be disabled using the *nx_dhcp_interface_disable* service discussed below).

At any time, an interface can be disabled for DHCP or DHCP can be stopped on that interface independently of other interfaces running DHCP.

As mentioned above, to enable a specific interface for DHCP, use the *nx_dhcp_interface_enable* service and specify the physical interface index in the input argument. Up to NX_DHCP_CLIENT_MAX_RECORDS interfaces can be enabled with the only limitation that the interface index input argument be less than NX_MAX_PHYSICAL_INTERFACES.

To start DHCP on a specific interface, use the *nx_dhcp_interface_start* service. To start DHCP on all enabled interfaces, use the *nx_dhcp_start* service. (Interfaces that have already started DHCP will not be affected by *nx_dhcp_start*.)

To stop DHCP on an interface, use the *nx_dhcp_interface_stop* service. DHCP must already have started on that interface or an error status is returned. To stop DHCP on all enabled interfaces, use the *nx_dhcp_stop* service. DHCP can be stopped independently of other interfaces at any time.

Most of the existing DHCP Client services have an ‘interface’ equivalent e.g. *nx_dhcp_interface_release* is the interface specific equivalent of *nx_dhcp_release.* If DHCP Client is configured for a single interface, they perform the same action.

Note that non-interface specific DHCP Client services typically apply to all interfaces but not all. In the latter case, the non-interface specific service applies to the first DHCP enabled interface found in searching the DHCP Client list of interface records. See **Description of Services** in Chapter Three for how a non-interface specific service performs when multiple interfaces are enabled for DHCP.

In the example sequence below, the IP instance has two network interfaces and first runs DHCP on the secondary interface. At some time later, it starts DHCP on the primary interface. Then it releases the IP address on the primary interface and restarts DHCP on the primary interface:

```
nx_dhcp_create(&my_dhcp_client);
/* By default this enables primary interface for DHCP.  */

nx_dhcp_interface_enable(&my_dhcp_client, 1);
/* Secondary interface is enabled. */

nx_dhcp_interface_start(&my_dhcp_client, 1);
/* DHCP is started on secondary interface. */

/* Some time later… */

nx_dhcp_interface_start(&my_dhcp_client, 0);
/* DHCP is started on primary interface. */

nx_dhcp_interface_release(&my_dhcp_client, 0);

/* Some time later… */

nx_dhcp_interface_start(&my_dhcp_client, 0);
/* DHCP is restarted on primary interface. */
```

For a complete list of interface specific services see **Description of Services** in Chapter Three.

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_DHCP_ENABLE_BOOTP	Defined, this option enables the BOOTP protocol instead of DHCP. By default this option is disabled.
NX_DHCP_CLIENT_RESTORE_STATE	If defined, this enables the DHCP Client to save its current DHCP Client license 'state' including time remaining on the lease, and restore this state between DHCP Client application reboots. The default value is disabled.
NX_DHCP_CLIENT_USER_CREATE_PACKET_POOL	If set, the DHCP Client will not create its own packet pool. The host application must use the nx_dhcp_packet_pool_set service to set the DHCP Client packet pool. The default value is disabled.
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_ARP_PROBE_WAIT	Defines the length of time the DHCP Client waits for a response after sending an ARP probe. The default value is one second (1 * NX_IP_PERIODIC_RATE)

NX_DHCP_ARP_PROBE_MIN	Defines the minimum variation in the interval between sending ARP probes. The value is defaulted to 1 second.
NX_DHCP_ARP_PROBE_MAX	Defines the maximum variation in the interval between sending ARP probes. The value is defaulted to 2 seconds.
NX_DHCP_ARP_PROBE_NUM	Defines the number of ARP probes sent for determining if the IP address assigned by the DHCP server is already in use. The value is defaulted to 3 probes.
NX_DHCP_RESTART_WAIT	Defines the length of time the DHCP Client waits to restart DHCP if the IP address assigned to the DHCP Client is already in use. The value is defaulted to 10 seconds.
NX_DHCP_CLIENT_MAX_RECORDS	Specifies the maximum number of interface records to save to the DHCP Client instance. A DHCP Client interface record is a record of the DHCP Client running on a specific interface. The default value is set as physical interfaces count(NX_MAX_PHYSICAL_INTERFACES).
NX_DHCP_CLIENT_SEND_MAX_DHCP_MESSAGE_OPTION	Defined, this enables the DHCP Client to send maximum DHCP message size option. By default, this option is disabled.
NX_DHCP_CLIENT_ENABLE_HOST_NAME_CHECK	Defined, this enables the DHCP Client to check the input host name in the nx_dhcp_create call for invalid characters or length. By default, this option is disabled.

NX_DHCP_THREAD_PRIORITY	Priority of the DHCP thread. By default, this value specifies that the DHCP thread runs at priority 3.
NX_DHCP_THREAD_STACK_SIZE	Size of the DHCP thread stack. By default, the size is 2048 bytes.
NX_DHCP_TIME_INTERVAL	Interval in seconds when the DHCP Client timer expiration function executes. This function updates all the timeouts in the DHCP process e.g. if messages should be retransmitted or DHCP Client state changed. By default, this value is 1 second.
NX_DHCP_OPTIONS_BUFFER_SIZE	Size of DHCP options buffer. By default, this value is 312.
NX_DHCP_PACKET_PAYLOAD	Specifies the size in bytes of the DHCP Client packet payload. The default value is NX_DHCP_MINIMUM_IP_DATAGRAM + physical header size. The physical header size in a wireline network is usually the Ethernet frame size.
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_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_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_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.
NX_DHCP_FRAGMENT_OPTION	Fragment enable for DHCP UDP requests. By default, this value is NX_DONT_FRAGMENT to disable DHCP UDP fragmenting.
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_QUEUE_DEPTH	Specifies the number of maximum depth of receive queue. The default value is set to 4.

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.

Retrieve DHCP Client's DHCP server address

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

Interface specific DHCP Client services:

nx_dhcp_interface_clear_broadcast_flag
Clear broadcast flag on Client messages
on specified interface

nx_dhcp_interface_enable
Enable interface to run DHCP
on the specified interface

nx_dhcp_interface_disable
Disable interface to run DHCP
on the specified interface

nx_dhcp_interface_decline
Send Decline message to server
on the specified interface

nx_dhcp_interface_force_renew
Send a force renew message
on the specified interface

nx_dhcp_interface_reinitialize
Clear DHCP client network parameters
on the specified interface

nx_dhcp_interface_release
Send Release message to server on the specified interface

nx_dhcp_interface_request_client_ip
Specify a specific IP address on the specified interface

nx_dhcp_interface_send_request
Send DHCP message to server on the specified interface

nx_dhcp_interface_server_address_get
Get the DHCP server IP address on the specified interface

nx_dhcp_interface_start
Start the DHCP Client processing on the specified interface

nx_dhcp_interface_stop
Stop the DHCP Client processing on the specified interface

nx_dhcp_interface_state_change_notify
Set the callback function when DHCP state changes on the specified interface

nx_dhcp_interface_user_option_retrieve
Retrieve the specified DHCP option on the specified interface

DHCP Client Services if NX_DHCP_CLIENT_RESORE_STATE is defined:

nx_dhcp_resume
Resume previously established DHCP Client state

nx_dhcp_suspend
Suspend processing the DHCP Client state

nx_dhcp_client_get_record
Create a record of the DHCP Client state

nx_dhcp_client_restore_record
Restore a previously saved record to the DHCP Client

nx_dhcp_client_update_time_remaining
Update the time remaining in the current DHCP state

Interface Specific DHCP Client Services if NX_DHCP_CLIENT_RESORE_STATE is defined:

nx_dhcp_client_interface_get_record
Create a record of the DHCP Client state on the specified interface

nx_dhcp_client_interface_restore_record
Restore a previously saved record to the DHCP Client on the specified interface

nx_dhcp_client_interface_update_time_remaining
Update the time remaining in the current DHCP state on the specified interface

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. By default the primary interface is enabled for running DHCP. The name input, while not used in the NetX implementation of DHCP Client, must follow RFC 1035 criteria for host names. The total length must not exceed 255 characters, the labels separate by dots must begin with a letter, and end with a letter or number, and may contain hyphens but no other non-alphanumeric character.

If the application would like to run DHCP another interface registered with the IP instance, (using *nx_ip_interface_attach*), the application can call *nx_dhcp_set_interface_index* to run DHCP on just that interface, or *nx_dhcp_interface_enable* to run DHCP on that interface as well. See description of these services for more details.

Note: The application must make sure the DHCP Client packet pool payload can support the minimum DHCP message size specified by the RFC 2131 Section 2 (548 bytes of DHCP message data plus UDP, IP and physical network frame headers).

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	Successful DHCP create
NX_DHCP_INVALID_NAME	(0xA8)	Invalid host name
NX_DHCP_INVALID_PAYLOAD	(0x9C)	Payload too small for DHCP message
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer

Allowed From

Threads, Initialization

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.  */
```

nx_dhcp_interface_enable

Enable the specified interface to run DHCP

Prototype

```
UINT nx_dhcp_interface_enable(NX_DHCP *dhcp_ptr, UINT interface_index);
```

Description

This service enables the specified interface for running DHCP. By default the primary interface is enabled for DHCP Client. At this point, DHCP can be started on this interface either by calling *nx_dhcp_interface_start* or to start DHCP on all enabled interfaces *nx_dhcp_start*.

Note the application must first register this interface with the IP instance, using *nx_ip_interface_attach.*

Further, there must be an available DHCP Client interface 'record' to add this interface to the list of enabled interfaces. By default `NX_DHCP_CLIENT_MAX_RECORDS` is defined to 1. Set this option to the maximum number of interfaces expected to run DHCP Client simultaneously. Typically `NX_DHCP_CLIENT_MAX_RECORDS` will equal NX`_MAX_PHYSICAL_INTERFACES`; however, if a device has more physical interfaces than it expects to run DHCP Client, it can save memory by setting `NX_DHCP_CLIENT_MAX_RECORDS` to less than that number. There is not a one to one mapping of physical interfaces with DHCP Client interface records.

The difference between this service and *nx_dhcp_set_interface_index* is the latter sets only a single interface to run DHCP whereas this service simply adds the specified interface to the list of Client interfaces enabled for DHCP.

To disable an interface for DHCP, the application can call the *nx_dhcp_interface_disable* service.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
interface_index	Index of interface to enable DHCP on

Return Values

NX_SUCCESS	(0x00)	Successful DHCP enable
NX_DHCP_NO_RECORDS_AVAILABLE	(0xA7)	No record available for another

		Interface to be enabled for DHCP
NX_DHCP_INTERFACE_ALREADY_ENABLED	(0xA3)	Interface enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads, Initialization

Example

```
/* Enable DHCP on a secondary interface. It is already enabled on the primary
   interface.  NX_DHCP_CLIENT_MAX_RECORDS is set to 2. */

status =  nx_dhcp_interface_enable(&my_dhcp, 1);
/* If status is NX_SUCCESS the interface was successfully enabled.  */


status = nx_dhcp_start(&my_dhcp);
/* If status is NX_SUCCESS DHCP is running on interface 0 and 1.  */
```

nx_dhcp_interface_disable

Disable the specified interface to run DHCP

Prototype

```
UINT nx_dhcp_interface_disable(NX_DHCP *dhcp_ptr,
                               UINT interface_index);
```

Description

This service disables the specified interface for running DHCP. It reinitializes the DHCP Client on this interface.

To restart the DHCP Client the application must re-enable the interface using *nx_dhcp_interface_enable* and restart DHCP by calling *nx_dhcp_interface_start*.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
interface_index	Index of interface to disable DHCP on

Return Values

NX_SUCCESS	(0x00)	Successful DHCP create
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Disable DHCP on a secondary interface.
. */

status =  nx_dhcp_interface_disable(&my_dhcp, 1);
/* If status is NX_SUCCESS the interface is successfully disabled.  */
```

nx_dhcp_clear_broadcast_flag

Set the DHCP broadcast flag

Prototype

```
UINT nx_dhcp_clear_broadcast_flag(NX_DHCP *dhcp_ptr, UINT clear_flag);
```

Description

This service sets or clears the broadcast flag the DHCP message header for all interfaces enabled for DHCP. For some DHCP messages (e.g. DISCOVER) the broadcast flag is set to broadcast because the Client does not have an IP address.

clear_flag	
NX_TRUE	broadcast flag is cleared (request unicast response)
NX_FALSE	broadcast flag is set (request broadcast response)

This service is intended for DHCP Clients that must go through a router to get to the DHCP Server, where the router rejects forwarding broadcast messages.

Input Parameters

dhcp_ptr	Pointer to DHCP control block
clear_flag	Value to set the broadcast flag to

Return Values

NX_SUCCESS	(0x00)	Successfully updated the broadcast flag
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads, Initialization

Example

```
/* Send DHCP Client messages with the broadcast flag cleared (e.g. request a
    unicast response).  */
status =  nx_dhcp_clear_broadcast_flag(&my_dhcp, NX_TRUE);

/* If status is NX_SUCCESS the DHCP Client broadcast flag is updated.  */
```

nx_dhcp_interface_clear_broadcast_flag

Set or clear the broadcast flag on the specified interface

Prototype

```
UINT nx_dhcp_interface_clear_broadcast_flag(NX_DHCP *dhcp_ptr,
                                            UINT interface_index,
                                            UINT clear_flag);
```

Description

This service enables the DHCP Client host application to set or clear the broadcast flag in DHCP Client messages to the DHCP Server on the specified interface. For more details see **nx_dhcp_clear_broadcast_flag**

Input Parameters

dhcp_ptr	Pointer to DHCP control block
interface_index	Index of interface to set the broadcast flag
clear_flag	Value to set the broadcast flag to

Return Values

NX_SUCCESS	(0x00)	Successfully updated the broadcast flag
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads, Initialization

Example

```
/* Send DHCP Client messages with the broadcast flag cleared (e.g. request a
   unicast response) on a previously attached secondary interface.  */

iface_index = 1;

status =  nx_dhcp_interface_clear_broadcast_flag(&my_dhcp, iface_index, NX_TRUE);

/* If status is NX_SUCCESS the DHCP Client broadcast flag is updated.  */
```

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.  */
```

nx_dhcp_ force_renew

Send a force renew message

Prototype

```
UINT nx_dhcp force_renew(NX_DHCP *dhcp_ptr);
```

Description

This service enables the host application to send a force renew message on all interfaces enabled for DHCP. The DHCP Client must be in a BOUND state. This function sets the state to RENEW such that the DHCP Client will try to renew before the T1 timeout expires.

To send a force renew on a specific interface when multiple interfaces are DHCP-enabled, use *nx_dhcp_interface_force_renew*.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successfully sent force renew.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Send a force renew message from the Client.  */
status =  nx_dhcp_force_renew(&my_dhcp);

/* If status is NX_SUCCESS the DHCP client state is the RENEWING state and the
   DHCP Client thread task will begin renewing before T1 is expired.  */
```

nx_dhcp_interface_force_renew

Send a force renew message on the specified interface

Prototype

```
UINT nx_dhcp_interface_force_renew(NX_DHCP *dhcp_ptr,
                                   UINT interface_index);
```

Description

This service enables the host application to send a force renew message on the input interface as long as that interface has been enabled for DHCP (see *nx_dhcp_interface_enable*). The DHCP Client on the specified interface must be in a BOUND state. This function sets the state to RENEW such that the DHCP Client will try to renew before the T1 timeout expires.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successfully sent force renew.
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Send a force renew message to the server on interface 1.  */
status =  nx_dhcp_interface_force_renew(&my_dhcp, 1);

/* If status is NX_SUCCESS the DHCP client state is the RENEWING state and the
   DHCP Client thread task will begin renewing before T1 is expired.  */
```

nx_dhcp_packet_pool_set

Set the DHCP Client packet pool

Prototype

```
UINT nx_dhcp_packet_pool_set(NX_DHCP *dhcp_ptr,
                              NX_PACKET_POOL *packet_pool_ptr);
```

Description

This service allows the application to create the DHCP Client packet pool by passing in a pointer to a previously created packet pool in this service call. To use this feature, the host application must define NX_DHCP_CLIENT_USER_CREATE_PACKET_POOL. When defined, the *nx_dhcp_create* service will not create the Client's packet pool. Note that the application is recommended to use the default values for the DHCP client packet pool payload, defined as NX_DHCP_PACKET_PAYLOAD in *nx_dhcp.h* when creating the packet pool.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
packet_pool_ptr	Pointer to previously created packet pool

Return Values

NX_SUCCESS	(0x00)	DHCP Client packet pool is set
NX_NOT_ENABLED	**(**0x14)	Service is not enabled
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer
NX_DHCP_INVALID_PAYLOAD	(0x9C)	Payload is too small

Allowed From

Application code

Example

```
/* Create the packet pool. */
status =  nx_packet_pool_create(&dhcp_pool, "DHCP Client Packet Pool",
       NX_DHCP_PACKET_PAYLOAD, pointer, (15 * NX_DHCP_PACKET_PAYLOAD));

/* Create the DHCP Client. */
status =  nx_dhcp_create(&dhcp_0, &ip_0, "janetsdhcp1");

/* Set the DHCP Client packet pool.  */
status =  nx_dhcp_packet_pool_set(&my_dhcp, packet_pool_ptr);
/* If status is NX_SUCCESS packet pool was successfully set.  */
```

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 Client to request from the DHCP Server on the first interface enabled for DHCP in the DHCP Client record. If the *skip_discover_message* flag is set, the DHCP Client skips the Discover message and sends a Request message.

To set the request for a specific IP for DHCP messages on a specific interface, use the *nx_dhcp_interface_request_client_ip* service.

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 If false, it sends the Discover message.

Return Values

NX_SUCCESS	(0x00)	Requested IP address is set.
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

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.  */
```

nx_dhcp_interface_request_client_ip

Set requested IP address for DHCP instance on specified interface

Prototype

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

Description

This service sets the IP address for the DHCP Client to request from the DHCP Server on the specified interface, if that interface is enabled for DHCP (see *nx_dhcp_interface_enable*). 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.
Interface_index	Index of interface to request IP address on
client_ip_address	IP address to request from DHCP server
skip_discover_message	If true, DHCP Client sends Request message; else it sends the Discover message.

Return Values

NX_SUCCESS	(0x00)	Requested IP address is set.
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid IP or DHCP pointer
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Set the DHCP Client requested IP address and skip the discover message on
   interface 0.  */
status =  nx_dhcp_interface_request_client_ip(&my_dhcp, 0, IP(192,168,0,6),
NX_TRUE);

/* If status is NX_SUCCESS requested IP address was successfully set.  */
```

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 clears the DHCP Client state on all interfaces enabled for DHCP. It is used in combination with *nx_dhcp_stop* and *nx_dhcp_start* to 'restart' the DHCP state machine:

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

To reinitialize the DHCP Client on a specific interface when multiple interfaces are enabled for DHCP, use the *nx_dhcp_interface_reinitialize* service.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	DHCP successfully reinitialized
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. */
```

nx_dhcp_interface_reinitialize

Clear the DHCP client network parameters on the specified interface

Prototype

```
UINT nx_dhcp_interface_reinitialize(NX_DHCP *dhcp_ptr,
                                    UINT interface_index);
```

Description

This service clears the network parameters (IP address, network address and network mask) on the specified interface if that interface is enabled for DHCP (see *nx_dhcp_interface_enable*). See *nx_dhcp_reinitialize* for more details.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance
interface_index	Index of interface to reinitialize.

Return Values

NX_SUCCESS	(0x00)	Interface successfully reinitialized
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Reinitialize the previously started DHCP client on interface 1.  */
status =  nx_dhcp_interface_reinitialize(&my_dhcp, 1);

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

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 a DHCP server by sending the RELEASE message to that server. It then reinitializes the DHCP Client. This service is applied to all interfaces enabled for DHCP.

The application can restart the DHCP Client by calling *nx_dhcp_start*.

To release an address back to the DHCP server on a specific interface, use the *nx_dhcp_interface_release* service

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

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

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

nx_dhcp_interface_release

Release IP address on the specified interface

Prototype

```
UINT nx_dhcp_interface_release(NX_DHCP *dhcp_ptr,
                               UINT interface_index);
```

Description

This service releases the IP address obtained from a DHCP server on the specified interface and reinitializes the DHCP Client. The DHCP Client can be restarted by calling *nx_dhcp_start*.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP release.
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
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.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Release the previously leased IP address on interface 1.  */
status =  nx_dhcp_interface_release(&my_dhcp, 1);

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

nx_dhcp_decline

Decline IP address from DHCP Server

Prototype

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

Description

This service declines an IP address leased from the DHCP server on all interfaces enabled for DHCP. If NX_DHCP_CLIENT_ SEND_ ARP_PROBE is defined, the DHCP Client will send a DECLINE message if it detects that the IP address is already in use. See **ARP Probes** in Chapter One for more information on ARP probe configuration in the NetX DHCP Client.

The application can use this service to decline its IP address if it discovers the address is in use by other means.

This service reinitializes the DHCP Client to that it can be restarted by calling *nx_dhcp_start*.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Decline successfully sent
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

```
/* Decline the IP address offered by the DHCP server.  */
status =  nx_dhcp_decline(&my_dhcp);

/* If status is NX_SUCCESS the previous IP address decline message was
successfully trasnmitted.  */
```

nx_dhcp_interface_decline

Decline IP address from DHCP Server on the specified interface

Prototype

```
UINT nx_dhcp_interface_decline(NX_DHCP *dhcp_ptr,
                               UINT interface_index);
```

Description

This service sends the DECLINE message to the server to decline an IP address assigned by the DHCP server. It also reinitializes the DHCP Client. See *nx_dhcp_decline* for more details.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.
Interface_index	Index of interface to decline IP address

Return Values

NX_SUCCESS	(0x00)	DHCP decline message sent
NX_DHCP_NOT_BOUND	(0x94)	DHCP Client not bound
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Decline the IP address offered by the DHCP server on interface 2.  */
status =  nx_dhcp_interface_decline(&my_dhcp, 2);

/* If status is NX_SUCCESS the previous IP address decline message was
successfully trasnmitted.  */
```

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 the specified DHCP message to the DHCP server on the first interface enabled for DHCP found in the DHCP Client record. To send a RELEASE or DECLINE message, the application must use the *nx_dhcp[_interface]_release*() or *nx_dhcp_interface_decline()* services respectively.

The DHCP Client must be started to use this service except for sending the INFORM_REQUEST message type.

Note: This service is not intended for the host application to 'drive' the DHCP Client state machine.

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	DHCP message sent
NX_DHCP_NOT_STARTED	(0x96)	Invalid interface index
NX_DHCP_INVALID_MESSAGE	(0x9B)	Invalid message type to send
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Send the DHCP INFORM REQUEST message to the server.  */

status =  nx_dhcp_send_request(&my_dhcp, NX_DHCP_TYPE_DHCPINFORM);
/* If status is NX_SUCCESS a DHCP message was successfully sent.  */
```

nx_dhcp_interface_send_request

Send DHCP message to Server on a specific interface

Prototype

```
UINT nx_dhcp_interface_send_request(NX_DHCP *dhcp_ptr,
                                    UINT interface_index,
                                    UINT dhcp_message_type);
```

Description

This service sends a message to the DHCP server on the specified interface if that interface is enabled for DHCP. To send a RELEASE or DECLINE message, the application must use the *nx_dhcp[_interface]_release*() or *nx_dhcp_interface_decline()* services respectively.

The DHCP Client must be started to use this service except for sending the DHCP INFORM REQUEST message type.

This service is not intended for the host application to 'drive' the DHCP Client state machine.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
Interface_index	Index of interface to send message on
dhcp_message_type	Message request (defined in *nx_dhcp.h*)

Return Values

NX_SUCCESS	(0x00)	DHCP message sent
NX_DHCP_NOT_STARTED	(0x96)	Invalid interface index
NX_DHCP_INVALID_MESSAGE	(0x9B)	Invalid message type to send
NX_DHCP_INTERFACE_NOT_ENABLED	(0xA4)	Interface not enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Send the INFORM REQUEST message to the server on the primary interface.  */

status =  nx_dhcp_interface_send_request(&my_dhcp, 0, NX_DHCP_TYPE_DHCPINFORM);
/* If status is NX_SUCCESS a DHCP message was successfully sent.  */
```

nx_dhcp_server_address_get

Get the DHCP Client's DHCP server IP address

Prototype

```
UINT nx_dhcp_server_address_get(NX_DHCP *dhcp_ptr,
                                ULONG server_address);
```

Description

This service retrieves the DHCP Client DHCP server IP address on the first interface enabled for DHCP found in the DHCP Client record. The caller can only use this service after the DHCP Client is bound to an IP address assigned by the DHCP Server. The host application can use the *nx_ip_status_check* service to verify IP address is set, or it can use the nx_*dhcp_state_change_notify* and query the DHCP Client state is NX_DHCP_STATE_BOUND. See *nx_dhcp_state_change_notify* for more details about setting the state change callback function.

To find the DHCP server on a specific interface when multiple interfaces are enabled for DHCP Client, use the *nx_dhcp_interface_server_address_get* service

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
server_address	Pointer to server IP address

Return Values

NX_SUCCESS	(0x00)	DHCP server address returned
NX_PTR_ERROR	(0x16)	Invalid input pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Use the state change notify service to determine the Client transition to the
bound state and get its DHCP server IP address.
/* void dhcp_state_change(NX_DHCP *dhcp_ptr, UCHAR new_state)
{

ULONG server_address;
UINT  status;
```

```
/* Increment state changes counter.  */
   state_changes++;

   if (dhcp_0.nx_dhcp_state == NX_DHCP_STATE_BOUND)
   {
          status = nx_dhcp_server_address_get(&dhcp_0, &server_address);
   }
```

nx_dhcp_interface_server_address_get

Get the DHCP Client's DHCP server IP address on the specified interface

Prototype

```
UINT nx_dhcp_interface_server_address_get(NX_DHCP *dhcp_ptr,
                                          UINT interface_index,
                                          ULONG server_address);
```

Description

This service retrieves the DHCP Client DHCP server IP address on the specified interface if that interface is enabled for DHCP. The DHCP Client must be in the Bound state. After starting the DHCP Client on that interface, the host application can either use the *nx_ip_status_check* service to verify the IP address is set, or it can use the DHCP Client state change callback and query the DHCP Client state is NX_DHCP_STATE_BOUND. See *nx_dhcp_state_change_notify* for more details about setting the state change callback function.

Input Parameters

dhcp_ptr	Pointer to DHCP control block.
Interface_index	Index of interface to obtain IP address
server_address	Pointer to server IP address

Return Values

NX_SUCCESS	(0x00)	DHCP server address returned
NX_DHCP_NO_INTERFACES_ENABLED	(0xA5)	No interfaces enabled for DHCP
NX_DHCP_NOT_BOUND	(0x94)	DHCP Client not bound
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Use the state change notify service to determine the Client transition to the
bound state and get its DHCP server IP address.
/* void dhcp_state_change(NX_DHCP *dhcp_ptr, UCHAR new_state)
{

ULONG server_address;
UINT  status;
```

```
/* Increment state changes counter.  */
   state_changes++;

   /* Get the DHCP server IP address on interface 1 */
   if (dhcp_0.nx_dhcp_state == NX_DHCP_STATE_BOUND)
   {
          status = nx_dhcp_interface_server_address_get(&dhcp_0, 1,
                                                 &server_address);
   }
```

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 for the DHCP instance to connect to the DHCP Server on when running DHCP Client configured for a single network interface.

By default the DHCP Client runs on the primary interface. To run DHCP on a secondary service, use this service to set the secondary interface as the DHCP Client interface. The application must previously register the specified interface to the IP instance using the *nx_ip_interface_attach* service.

Note that this service is intended for applications that intend to run the DHCP Client on only one interface. To run DHCP on multiple interfaces see *nx_dhcp_interface_enable* for more details.

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_INVALID_INTERFACE	(0x4C)	Invalid network interface
NX_DHCP_INTERFACE_ALREADY_ENABLED	(0xA3)	Interface enabled for DHCP
NX_DHCP_NO_RECORDS_AVAILABLE	(0xA7)	No record available for another
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.  */
```

nx_dhcp_start

Start DHCP processing

Prototype

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

Description

This service starts DHCP processing on all interfaces enabled for DHCP. By default the primary interface is enabled for DHCP when the application calls *nx_dhcp_create.*

To verify when the IP instance is bound to an IP address on the DHCP Client interface, use *nx_ip_status_check* to see confirm the IP address is valid.

If there are other interfaces already running DHCP, this service will not affect them.

To start DHCP on a specific interface when multiple interfaces are enabled, use the *nx_dhcp_interface_start* service.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP start.
NX_DHCP_ALREADY_STARTED	(0x93)	DHCP already 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.  */
```

nx_dhcp_interface_start

Start DHCP processing on the specified interface

Prototype

```
UINT nx_dhcp_interface_start(NX_DHCP *dhcp_ptr, UINT interface_index);
```

Description

This service starts DHCP processing on the specified interface if that interface is enabled for DHCP. See *nx_dhcp_interface_enable*() for more details about enabling an interface for DHCP. By default the primary interface is enabled for DHCP when the application calls *nx_dhcp_create.*

If there are no other interfaces running DHCP Client this service will start/resume the DHCP Client thread and (re)activate the DHCP Client timer.

The application should use *nx_ip_status_check* to verify if an IP address is obtained.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.
Interface_index	Index on which to start the DHCP Client

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.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Start the DHCP processing for this IP instance on interface 1.  */
status =  nx_dhcp_interface_start(&my_dhcp, 1);

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

nx_dhcp_state_change_notify

Set DHCP state change callback function

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 callback function `dhcp_state_change_notify` for notifying an application of DHCP state changes. The callback function supplies the state the DHCP Client has transitioned into.

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
NX_DHCP_STATE_ADDRESS_PROBING	9

Input Parameters

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

Return Values

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

Allowed From

Threads, Initialization

Example

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

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

nx_dhcp_interface_state_change_notify

Set DHCP state change callback function on the specified interface

Prototype

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

Description

This service registers the specified callback function for notifying an application of DHCP state changes. The callback funciton input arguments are the interface index and the state the DHCP Client has transitioned to on that interface.

For more information about state change functions, see *nx_dhcp_state_change_notify*().

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	Successful callback set.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.

Allowed From

Threads, Initialization

Example

```
/* Register the “my_state_change” function to be called on any DHCP state change,
   assuming DHCP has alreadybeen created.  */

void dhcp_interstate_state_change(NX_DHCP *dhcp_ptr, UINT iface_index,
                                  UCHAR new_state);


status =  nx_dhcp_interstate_state_change_notify(&my_dhcp,
                                                  dhcp_interstate_state_change);
```

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

nx_dhcp_stop

Stops DHCP processing

Prototype

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

Description

This service stops DHCP processing on all interfaces that have started DHCP processing. If there are no interfaces processing DHCP, this service will suspend the DHCP Client thread, and inactivate the DHCP Client timer.

To stop DHCP on a specific interface if multiple interfaces are enabled for DHCP, use the *nx_dhcp_interface_stop* service.

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.  */
```

nx_dhcp_interface_stop

Stop DHCP processing on the specified interface

Prototype

```
UINT nx_dhcp_interface_stop(NX_DHCP *dhcp_ptr, UINT interface_index);
```

Description

This service stops DHCP processing on the specified interface if DHCP is already started. If there are no other interfaces running DHCP, the DHCP thread and timer are suspended.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.
Interface_index	Interface on which to stop DHCP processing

Return Values

NX_SUCCESS	(0x00)	Successful DHCP stop
NX_DHCP_NOT_STARTED	(0x96)	DHCP not started.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of service.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
/* Stop DHCP processing for this IP instance on interface 1.  */
status =  nx_dhcp_interface_stop(&my_dhcp, 1);

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

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 DHCP options buffer on the first interface enabled for DHCP found on the DHCP Client record. If successful, the option data is copied into the specified 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` option 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 option retrieval.
NX_DHCP_NOT_BOUND	(0x94)	DHCP Client not bound.
NX_DHCP_NO_INTERFACES_ENABLED	**(0xA5)**	No interfaces enabled for DHCP
NX_DHCP_DEST_TO_SMALL	(0x95)	Destination is too small to hold response.
NX_DHCP_PARSE_ERROR	(0x97)	DHCP Option not found in Server response.
NX_PTR_ERROR	(0x16)	Invalid input 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.  */
```

nx_dhcp_interface_user_option_retrieve

Retrieve a DHCP option from last server response on the specified interface

Prototype

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

Description

This service retrieves the specified DHCP option from the DHCP options buffer on the specified interface, if that interface is enabled for DHCP. If successful, the option data is copied into the specified buffer.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.
Interface_index	Index on which to retrieve the specified option
request_option	DHCP option, as specified by the RFCs. See the `NX_DHCP_OPTION` option 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 option retrieval.
NX_DHCP_NOT_BOUND	(0x94)	IP address not assigned
NX_DHCP_DEST_TO_SMALL	(0x95)	Buffer is too small
NX_DHCP_PARSE_ERROR	(0x97)	DHCP Option not found in Server response.
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of service.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

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

/* Obtain the IP address of the DNS server on the prmary interface.  */
size =    sizeof(dnx_ip_string);
```

```
status =  nx_dhcp_interface_user_option_retrieve(&my_dhcp, 0,
NX_DHCP_OPTION_DNS_SVR,
               dns_ip_string, &size);

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

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.”  */
```

nx_dhcp_user_option_add_callback_set

Set callback function for adding user supplied options

Prototype

```
ULONG nx_dhcp_user_option_add_callbcak_set(NX_DHCP *dhcp_ptr,
      UINT (*dhcp_user_option_add)(NX_DHCP *dhcp_ptr,
                                   UINT iface_index,
                                   UINT message_type,
                                   UCHAR *user_option_ptr,
                                   UINT *user_option_length));
```

Description

This service registers the specified callback function for adding user supplied options.

If the callback function specified, the applications can add user supplied options into the packet by iface_index and message_type.

Note: In user's routine. Applications must follow the DHCP options format when add user supplied options. The total size of user options must be less or equal to user_option_length, and update the user_option_length as real options length. Return NX_TRUE if add options successfully, else return NX_FALSE.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.
dhcp_user_option_add	Pointer to user option add function.

Return Values

NX_SUCCESS	(0x00)	Successful callback set.
NX_PTR_ERROR	(0x16)	Invalid pointer.

Allowed From

Threads

Example

```
/* Register the "my_dhcp_user_option_add" function to be called when add DHCP
   options, assuming DHCP has already been created.  */

status =  nx_dhcp_user_option_add_callback_set(&my_dhcp, my_dhcp_user_option_add);

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

Appendix A

Description of the Restore State Feature

The NetX DHDP Client configuration option, NX_DHCP_CLIENT_RESTORE_STATE, allows a system to restore a previously created DHCP Client Record in a Bound state between system reboots.

When this option is enabled, the application can suspend and resume the DHCP Client thread. There is also a service to update the DHCP Client with the elapsed time between suspending and resuming the thread.

Restoring the DHCP Client between Reboots

Before restoring a DHCP Client after rebooting, a previously created DHCP Client that must reach the Bound state and be is assigned an IP address from the DHCP server. Before it powers down, the DHCP application must then save the current DHCP Client record to non-volatile memory. There must also be an independent 'time keeper' elsewhere in the system to keep track of the time elapsed during this powered down state. On powering up, the application creates a new DHCP Client instance, and then updates it with the previously created DHCP Client record. The elapsed time is obtained from the "time keeper" and then applied to the time remaining on the DHCP Client lease. Note that this may cause the DHCP Client to change states e.g. from BOUND to RENEWING. At this point, the application can resume the DHCP Client.

If the time elapsed during power down puts the DHCP Client state in either a RENEW or REBIND state, the DHCP Client will automatically initiate DHCP messages requesting to renew or rebind the IP address lease. If the IP address is expired, the DHCP Client will automatically clear the IP address on the IP instance and begin the DHCP process from the INIT state, requesting a new IP address.

In this manner the DHCP Client can operate between reboots as if uninterrupted.

Below is an illustration of this feature. This assumes DHCP Client is running only on the primary interface.

```
/* On the power up, create an IP instance, DHCP Client, enable ICMP and UDP
   and other resources (not shown) for the DHCP Client/application
   in tx_application_define().  */


/* Define the DHCP application thread. */
void    thread_dhcp_client_entry(ULONG thread_input)
{
```

```
UINT        status;
UINT        time_elapsed = 0;
NX_DHCP_CLIENT_RECORD client_nv_record;


if (/* The application checks if there is a previously saved DHCP Client record. */)
{


    /* No previously saved Client record. Start the DHCP Client in the INIT state.  */
    status =  nx_dhcp_start(&dhcp_0);

    if (status !=NX_SUCCESS)
        return;

    do
    {

        /* Wait for DHCP to assign the IP address.  */
    } while (status != NX_SUCCESS);

    /* We have a valid IP address. */

    /* At some point decide we power down the system. */

    /* Save the Client state data which we will subsequently need to restore the DHCP
       Client. */
    status = nx_dhcp_client_get_record(&dhcp_0, &client_nv_record);

    /* Copy this memory to non-volatile memory (not shown). */

    /* Delete the IP and DHCP Client instances before powering down. */
    nx_dhcp_delete(&dhcp_0);

    nx_ip_delete(&ip_0);

    /* Ready to power down, having released other resources as necessary. */

}
else
{

      /* The application has determined there is a previously saved record. We will
         restore it to the current DHCP Client instance.  */

      /* Get the previous Client state data from non-volatile memory. */

      /* Apply the record to the current Client instance. This will also
         update the IP instance with IP address, mask etc. */
      status = nx_dhcp_client_restore_record(&dhcp_0, &client_nv_record, time_elapsed);

     if (status != NX_SUCCESS)
          return;

     /* We are ready to resume the DHCP Client thread and use the assigned IP address. */
     status = nx_dhcp_resume(&dhcp_0);

     if (status != NX_SUCCESS)
          return;

}
```

Resuming the DHCP Client Thread after Suspension

To suspend a DHCP Client thread without powering down, the application calls *nx_dhcp_suspend* on a DHCP Client which has achieved the BOUND state and which has a valid IP address. When it is ready to resume the DHCP Client it first calls *nx_dhcp_client_update_time_remaining* to update the time remaining on the DHCP address lease (obtaining the time elapsed from an independent time keeper). Then it calls the *nx_dhcp_resume* to resume the DHCP Client thread.

If the time elapsed puts the DHCP Client state in either a RENEW or REBIND state, the DHCP Client will automatically initiate DHCP messages requesting to renew or rebind the IP address lease. If the IP address is expired, the DHCP Client will automatically clear the IP address and begin the DHCP process from the INIT state, requesting a new IP address.

Below is an illustration of using this feature.

```
/* Create an IP instance, DHCP Client, enable ICMP and UDP
   and other resources (not shown) typically in tx_application_define().  */


/* Define the DHCP application thread. */
void    thread_dhcp_client_entry(ULONG thread_input)
{


  /* Start the DHCP Client.  */
  status =  nx_dhcp_start(&dhcp_0);

  if (status !=NX_SUCCESS)
    return;

  while(1)
  {

       /* Wait for DHCP to obtain an IP address.  */
  }

  /* Do tasks with the IP address e.g. send pings to another host on the network...  */
  status =  nx_icmp_ping(…);

  if (status !=NX_SUCCESS)
          printf("Failed %d byte Ping!\n", length);

  /* At some later time, suspend the DHCP Client e.g. the device is going to low
   power mode (sleep) so we do not want any threads to wake it up. */

  nx_dhcp_suspend(&dhcp_0);

  /* During this suspended state, an independent timer is keeping track of the elapsed
     time. */


  /* At some point, we are ready to resume the DHCP Client thread. */

  /* Update the DHCP Client lease time remaining with the time elapsed. */
  status = nx_dhcp_client_update_time_remaining(&dhcp_0, time_elapsed);

  if (status != NX_SUCCESS)
       return;

  /* We now can resume the DHCP Client thread. */
  status = nx_dhcp_resume(&dhcp_0);

  if (status != NX_SUCCESS)
       return;

  /* Resume tasks e.g. ping another host. */
  status =  nx_icmp_ping(…);

}
```

Below is a list of services for restoring a DHCP Client state from memory and for suspending and resuming the DHCP Client.

nx_dhcp_client_get_record

Create a record of the current DHCP Client state

Prototype

```
ULONG nx_dhcp_ client_get_record(NX_DHCP *dhcp_ptr,
                                  NX_DHCP_CLIENT_RECORD *record_ptr);
```

Description

This service saves the DHCP Client running on the first interface enabled for DHCP found on the DHCP Client instance to the record pointed to by `record_ptr`. This allows the DHCP Client application restore its DHCP Client state after, for example, a power down and reboot.

To save a DHCP Client record on a specific interface if more than one interface is enabled for DHCP, use the *nx_dhcp_interface_client_get_record* service.

Input Parameters

dhcp_ptr	Pointer to DHCP Client
record_ptr	Pointer to DHCP Client record

Return Values

NX_SUCCESS	(0x0)	Client record created
NX_DHCP_NOT_BOUND	(0x94)	Client not in Bound state
NX_DHCP_NO_INTERFACES_ENABLED	(0xA5)	No interfaces enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
NX_DHCP_CLIENT_RECORD dhcp_record;


/* Obtain a record of the current client state. */
status=  nx_dhcp_client_get_record(dhcp_ptr, &dhcp_record);

/* If status is NX_SUCCESS dhcp_record contains the current DHCP client record. */
```

nx_dhcp_interface_client_get_record

Create a record of the current DHCP Client state on the specified interface

Prototype

```
ULONG nx_dhcp_interface_client_get_record(NX_DHCP *dhcp_ptr,
                                          UINT interface_index,
                                          NX_DHCP_CLIENT_RECORD *record_ptr);
```

Description

This service saves the DHCP Client running on the specified interface to the record pointed to by `record_ptr`. This allows the DHCP Client application restore its DHCP Client state after, for example, a power down and reboot.

Input Parameters

dhcp_ptr	Pointer to DHCP Client
interface_index	Index on which to get record
record_ptr	Pointer to DHCP Client record

Return Values

NX_SUCCESS	(0x0)	Client record created
NX_DHCP_NOT_BOUND	(0x94)	Client not in Bound state
NX_DHCP_BAD_INTERFACE_INDEX_ERROR	(0x9A)	Invalid interface index
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
NX_DHCP_CLIENT_RECORD dhcp_record;

/* Obtain a record of the current client state on interface 1. */
status=  nx_dhcp_interface_client_get_record(dhcp_ptr, 1, &dhcp_record);

/* If status is NX_SUCCESS dhcp_record contains the current DHCP client record. */
```

nx_dhcp_ client_restore_record

Restore DHCP Client from a previously saved record

Prototype

```
ULONG nx_dhcp_client_restore_record(NX_DHCP *dhcp_ptr,
                                    NX_DHCP_CLIENT_RECORD
                                    *record_ptr, ULONG time_elapsed);
```

Description

This service enables an application to restore its DHCP Client from a previous session using the DHCP Client record pointed to by `record_ptr`. `The time_elapsed` input is applied to the time remaining on DHCP Client lease.

This requires that the DHCP Client application created a record of the DHCP Client before powering down, and saved that record to nonvolatile memory.

If more than one interface is enabled for DHCP Client, this service is applied to the first valid interface found in the DHCP Client instance.

Input Parameters

dhcp_ptr	Pointer to DHCP Client
record_ptr	Pointer to DHCP Client record
time_elapsed	Time to subtract from the lease time remaining in the input client record

Return Values

NX_SUCCESS	(0x0)	Client record restored
NX_DHCP_NO_INTERFACES_ENABLED	(0xA5)	No interfaces running DHCP
NX_PTR_ERROR	(0x16)	Invalid pointer Input

Allowed From

Threads

Example

```
NX_DHCP_CLIENT_RECORD dhcp_record;
ULONG          time_elapsed;

/* Obtain time (timer ticks) elapsed from independent time keeper. */
```

```
Time_elapsed = /* to be determined by application */ 1000;


/* Obtain a record of the current client state. */
status=  nx_dhcp_client_restore_record(client_ptr, &dhcp_record, time_elapsed);

/* If status is NX_SUCCESS the current DHCP Client pointed to by dhcp_ptr
contains the current client record updated for time elapsed during power down. */
```

nx_dhcp_interace_client_restore_record

Restore DHCP Client from a previously saved record on specified interface

Prototype

```
ULONG nx_dhcp_interface_client_restore_record(NX_DHCP *dhcp_ptr,
                                    NX_DHCP_CLIENT_RECORD
                                    *record_ptr, ULONG time_elapsed);
```

Description

This service enables an application to restore its DHCP Client on the specified interface using the DHCP Client record pointed to by `record_ptr`. The `time_elapsed` input is applied to the time remaining on DHCP Client lease.

This requires that the DHCP Client application created a record of the DHCP Client before powering down, and saved that record to nonvolatile memory.

If more than one interface is enabled for DHCP Client, this service is applied to the first valid interface found in the DHCP Client instance.

Input Parameters

dhcp_ptr	Pointer to DHCP Client
record_ptr	Pointer to DHCP Client record
time_elapsed	Time to subtract from the lease time remaining in the input client record

Return Values

NX_SUCCESS	(0x0)	Client record restored
NX_DHCP_NOT_BOUND	(0x94)	Client not bound to IP address
NX_DHCP_BAD_INTERFACE_INDEX_ERROR	(0x9A)	Invalid interface index
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
NX_DHCP_CLIENT_RECORD dhcp_record;
```

```
ULONG          time_elapsed;

/* Obtain time (timer ticks) elapsed from independent time keeper. */
Time_elapsed = /* to be determined by application */ 1000;


/* Obtain a record of the current client state on the primary interface. */
status=  nx_dhcp_interface_client_restore_record(client_ptr, 0, &dhcp_record,
time_elapsed);

/* If status is NX_SUCCESS the current DHCP Client pointed to by dhcp_ptr
contains the current client record updated for time elapsed during power down. */
```

nx_dhcp_ client_update_time_remaining

Update the time remaining on DHCP Client lease

Prototype

```
ULONG nx_dhcp_client_update_time_remaining(NX_DHCP *dhcp_ptr
                                            ULONG time_elapsed);
```

Description

This service updates the time remaining on the DHCP Client IP address lease with the `time_elapsed` input on the first interface enabled for DHCP found on the DHCP Client instance. The application must suspend the DHCP Client thread before using this service using *nx_dhcp_suspend*. After calling this service, the application can resume the DHCP Client thread by calling *nx_dhcp_resume*.

This is intended for DHCP Client applications that need to suspend the DHCP Client thread for a period of time, and then update the IP address lease time remaining.

Note: This service is not intended to be used with *nx_dhcp_client_get_record* and *nx_dhcp_client_restore_record* described previously). These services are previously described in this section.

Input Parameters

dhcp_ptr	Pointer to DHCP Client
time_elapsed	Time to subtract from the time remaining on the IP address lease

Return Values

NX_SUCCESS	(0x0)	Client IP lease updated
NX_DHCP_NO_INTERFACES_ENABLED	(0xA5)	No interfaces enabled for DHCP
NX_PTR_ERROR	(0x16)	Invalid Pointer Input

Allowed From

Threads

Example

```
ULONG          time_elapsed;

/* Obtain time (timer ticks) elapsed from independent time keeper. */
```

```
time_elapsed = /* to be determined by application */ 1000;


/* Apply the elapsed time to the DHCP Client address lease. */
status=  nx_dhcp_client_update_time_remaining(client_ptr, time_elapsed);

/* If status is NX_SUCCESS the DHCP Client is updated for time elapsed. */
```

nx_dhcp_interface_client_update_time_remaining

Update the time remaining on DHCP Client lease on the specified interface

Prototype

```
ULONG nx_dhcp_interface_client_update_time_remaining(NX_DHCP *dhcp_ptr,
                                              UINT interface_index,
                                              ULONG time_elapsed);
```

Description

This service updates the time remaining on the DHCP Client IP address lease with the `time_elapsed` input on the specified interface if that interface is enabled for DHCP. The application must suspend the DHCP Client thread before using this service using *nx_dhcp_suspend*. After calling this service, the application can resume the DHCP Client thread by calling *nx_dhcp_resume*. Note suspending and resuming the DHCP Client thread applies to all interfaces enabled for DHCP.

This is intended for DHCP Client applications that need to suspend the DHCP Client thread for a period of time, and then update the IP address lease time remaining.

Note: This service is not intended to be used with *nx_dhcp_client_get_record* and *nx_dhcp_client_restore_record* described previously). These services are previously described in this section.

Input Parameters

dhcp_ptr	Pointer to DHCP Client
interface_index	Index to interface to apply elapsed time to
time_elapsed	Time to subtract from the time remaining on the IP address lease

Return Values

NX_SUCCESS	(0x0)	Client IP lease updated
NX_DHCP_BAD_INTERFACE_INDEX_ERROR	(0x9A)	Invalid interface index
NX_PTR_ERROR	(0x16)	Invalid DHCP pointer.
NX_INVALID_INTERFACE	(0x4C)	Invalid network interface

Allowed From

Threads

Example

```
ULONG          time_elapsed;

/* Obtain time (timer ticks) elapsed from independent time keeper. */
time_elapsed = /* to be determined by application */ 1000;


/* Apply the elapsed time to the DHCP Client address lease on interface 1. */
status=  nx_dhcp_interface_client_update_time_remaining(client_ptr, 1,
time_elapsed);

/* If status is NX_SUCCESS the DHCP Client is updated for time elapsed. */
```

nx_dhcp_suspend

Suspend the DHCP Client thread

Prototype

```
ULONG nx_dhcp_suspend(NX_DHCP *dhcp_ptr);
```

Description

This service suspends the current DHCP Client thread. Note that unlike *nx_dhcp_stop*, there is no change to the DHCP Client state when this service is called.

This service suspends DHCP running on all interfaces enabled for DHCP.

To update the DHCP Client state with elapsed time while the DHCP Client is suspended, see the *nx_dhcp_client_update_time_remaining* described previously. To resume a suspended DHCP Client thread, the application should call *nx_dhcp_resume*.

Input Parameters

dhcp_ptr	Pointer to DHCP Client

Return Values

NX_SUCCESS	(0x0)	Client thread is suspended
NX_PTR_ERROR	(0x16)	Invalid pointer Input

Allowed From

Threads

Example

```
/* Pause the DHCP client thread. */
status=  nx_dhcp_suspend(client_ptr);

/* If status is NX_SUCCESS the current DHCP Client thread is paused. */
```

nx_dhcp_resume

Resume a suspended DHCP Client thread

Prototype

```
ULONG nx_dhcp_resume(NX_DHCP *dhcp_ptr);
```

Description

This service resumes a suspended DHCP Client thread. Note that there is no change to the actual DHCP Client state after resuming the Client thread. To update the time remaining on the DHCP Client IP address lease with elapsed time before calling *nx_dhcp_resume*, see the *nx_dhcp_client_update_time_remaining* described previously.

This service resumes DHCP running on all interfaces enabled for DHCP.

Input Parameters

dhcp_ptr	Pointer to DHCP Client

Return Values

NX_SUCCESS	(0x0)	Client thread is resumed
NX_PTR_ERROR	(0x16)	Invalid pointer Input

Allowed From

Threads

Example

```
/* Resume the DHCP client thread. */
status=  nx_dhcp_resume(client_ptr);

/* If status is NX_SUCCESS the current DHCP Client thread is resumed. */
```