



## **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](http://www.expresslogic.com)

**©2002-2016 by Express Logic, Inc.**

All rights reserved. This document and the associated NetX software are the sole property of Express Logic, Inc. Each contains proprietary information of Express Logic, Inc. Reproduction or duplication by any means of any portion of this document without the prior written consent of Express Logic, Inc. is expressly forbidden. Express Logic, Inc. reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of NetX. The information in this document has been carefully checked for accuracy; however, Express Logic, Inc. makes no warranty pertaining to the correctness of this document.

**Trademarks**

NetX, Piconet, and UDP Fast Path are trademarks of Express Logic, Inc. ThreadX is a registered trademark of Express Logic, Inc.

All other product and company names are trademarks or registered trademarks of their respective holders.

**Warranty Limitations**

Express Logic, Inc. makes no warranty of any kind that the NetX products will meet the USER's requirements, or will operate in the manner specified by the USER, or that the operation of the NetX products will operate uninterrupted or error free, or that any defects that may exist in the NetX products will be corrected after the warranty period. Express Logic, Inc. makes no warranties of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with respect to the NetX products. No oral or written information or advice given by Express Logic, Inc., its dealers, distributors, agents, or employees shall create any other warranty or in any way increase the scope of this warranty, and licensee may not rely on any such information or advice.

Part Number: 000-1050

Revision 5.9

# Contents

---

Chapter 1 Introduction to DHCP Client.....	5
Dynamic IP Address Assignment .....	5
RARP Alternatives.....	5
DHCP Messages .....	6
DHCP Communication .....	6
DHCP Client State Machine .....	6
DHCP User Request .....	7
DHCP RFCs.....	7
Chapter 2 Installation and Use of DHCP Client .....	8
Product Distribution.....	8
DHCP Installation .....	8
Using DHCP .....	8
In the Bound State.....	9
Sending DHCP Messages To The Server .....	9
Starting and Stopping the DHCP Client .....	10
Using the DHCP Client with Auto IP .....	11
Small Example System .....	11
Multi-Server Environments .....	14
BOOTP Protocol.....	14
DHCP Multihome Support .....	14
Configuration Options .....	16
Chapter 3 Description of DHCP Client Services .....	21
nx_dhcp_create .....	23
nx_dhcp_clear_broadcast_flag .....	24
nx_dhcp_delete .....	25
nx_dhcp_decline .....	26
nx_dhcp_force_renew.....	27
nx_dhcp_packet_pool_set.....	28
nx_dhcp_reinitialize.....	29
nx_dhcp_release.....	30
nx_dhcp_request_client_ip .....	31
nx_dhcp_send_request.....	32
nx_dhcp_server_address_gset.....	33
nx_dhcp_set_interface_index .....	34
nx_dhcp_start.....	35
nx_dhcp_state_change_notify.....	36
nx_dhcp_stop .....	38
nx_dhcp_user_option_retrieve.....	39
nx_dhcp_user_option_convert.....	41
Appendix A - Description of the Restore State Feature .....	42
Restoring the DHCP Client between Reboots .....	42

Resuming the DHCP Client Thread after Suspension .....	43
nx_dhcp_client_get_record.....	45
nx_dhcp_client_restore_record.....	46
nx_dhcp_client_update_time_remaining.....	48
nx_dhcp_suspend.....	50
nx_dhcp_resume .....	51

# Chapter 1

## Introduction to DHCP Client

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

### **Dynamic IP Address Assignment**

---

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

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

### **RARP Alternatives**

---

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

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

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

### **DHCP Messages**

---

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

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

### **DHCP Communication**

---

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

### **DHCP Client State Machine**

---

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

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

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

### **DHCP User Request**

---

Once the DHCP server grants an IP address, the DHCP client processing can request additional parameters — one at a time — by using the *nx\_dhcp\_user\_option\_request* service.

### **DHCP RFCs**

---

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

## Chapter 2

# Installation and Use of DHCP Client

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

### Product Distribution

---

DHCP for NetX is shipped on a single CD-ROM compatible disk. The package includes two source files and a PDF file that contains this document, as follows:

<b><code>nx_dhcp.h</code></b>	Header file for DHCP for NetX
<b><code>nx_dhcp.c</code></b>	C Source file for DHCP for NetX
<b><code>nx_dhcp.pdf</code></b>	PDF description of DHCP for NetX
<b><code>demo_netx_dhcp.c</code></b>	NetX DHCP demonstration

### DHCP Installation

---

In order to use DHCP for NetX, the entire distribution mentioned previously should be copied to the same directory where NetX is installed. For example, if NetX is installed in the directory “`\threadx\arm7\green`” then the `nx_dhcp.h` and `nx_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 client thread iterates once per interval (as specified by `NX_DHCP_TIME_INTERVAL`). During this interval it will decrement the time remaining on the IP address lease. There is an option to periodically check for DHCP Server messages. This is set by the `NX_DHCP_TIMEOUT_DECREMENTS` option which determines the intervals between checking for messages as follows:

`NX_DHCP_TIMEOUT_DECREMENTS * NX_DHCP_TIME_INTERVAL`

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

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

When the time remaining reaches the T1 (renewal timeout), the DHCP Client is promoted to the RENEW state. It will then send requests renewing its IP lease from the DHCP Server. If the Server has not responded when the time remaining reaches T2 (rebind timeout), the DHCP Client is promoted to the REBIND state. It will then broadcast requests for a new IP address to any DHCP Server on the network. If no Server replies when the IP lease expires, the DHCP Client is reset back to the INIT state and it restarts the IP address request with the DISCOVER messages.

## **Sending DHCP Messages To The Server**

---

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

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

- *nx\_dhcp\_forcerenew*: this does not send a message but sets the DHCP Client in the FORCERENEW state if the Server sends the Client a FORCERENEW message. The DHCP Client will then set itself to the RENEW state to begin requesting IP lease renewal.
- *nx\_dhcp\_send\_request*: This takes as an argument a DHCP message type, as specified in *nx\_dhcp.h*, and sends the message to the Server. This is how a host application would send a DECLINE or INFORM\_REQUEST to the Server.

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

## Starting and Stopping the DHCP Client

---

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

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

A typical sequence might be:

```
nx_dhcp_stop(&my_dhcp);

nx_dhcp_reinitialize(&my_dhcp);

nx_dhcp_start(&my_dhcp);
```

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

## Using the DHCP Client with Auto IP

---

The NetX DHCP Client works concurrently with the Auto IP protocol in applications where DHCP and Auto IP guarantee an address where a DHCP Server is not guaranteed to be available or responding. However, if the host is unable to detect a Server or get an IP address assigned, it can switch to the Auto IP protocol for a local IP address. However before doing so, it is advisable to stop the DHCP Client temporarily while Auto IP goes through the “probe” and “defense” stages. Once an Auto IP address is assigned to the host, the DHCP Client can be restarted and if a DHCP Server does become available, the host IP address can accept the IP address offered by the DHCP Server while the application is running.

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

## Small Example System

---

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

```

0001 #include "tx_api.h"
0002 #include "nx_api.h"
0003 #include "nx_dhcp.h"
0004
0005 #define DEMO_STACK_SIZE 4096
0006 TX_THREAD my_thread;
0007 NX_PACKET_POOL my_pool;
0008 NX_IP my_ip;
0009 NX_DHCP my_dhcp;
0010
0011 /* Define function prototypes. */
0012
0013 void my_thread_entry(ULONG thread_input);
0014 void my_netx_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
0015
0016 /* Define main entry point. */
0017
0018 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     /* Setup the working pointer. */
0035     pointer = (CHAR *) first_unused_memory;
0036
0037     /* Create "my_thread". */
0038     tx_thread_create(&my_thread, "my thread", my_thread_entry, 0,
0039         pointer, DEMO_STACK_SIZE,
0040         2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
0041     pointer = pointer + DEMO_STACK_SIZE;
0042
0043     /* Initialize the NetX system. */
0044     nx_system_initialize();
0045
0046     /* Create a packet pool. */
0047     status = nx_packet_pool_create(&my_pool, "NetX Main Packet Pool",
0048         1024, pointer, 64000);
0049     pointer = pointer + 64000;
0050
0051     /* check for pool creation error. */
0052     if (status)
0053         error_counter++;
0054
0055     /* Create an IP instance without an IP address. */
0056     status = nx_ip_create(&my_ip, "My NetX IP Instance", IP_ADDRESS(0,0,0,0),
0057         0xFFFFFFFF, &my_pool, my_netx_driver, pointer,
0058         DEMO_STACK_SIZE, 1);
0059     pointer = pointer + DEMO_STACK_SIZE;
0060
0061     /* Check for IP create errors. */
0062     if (status)
0063         error_counter++;
0064
0065     /* Enable ARP and supply ARP cache memory for my IP Instance. */
0066     status = nx_arp_enable(&my_ip, (void *) pointer, 1024);
0067     pointer = pointer + 1024;
0068
0069     /* check for ARP enable errors. */
0070     if (status)
0071         error_counter++;
0072
0073     /* Enable UDP. */
0074     status = nx_udp_enable(&my_ip);
0075     if (status)
0076         error_counter++;
0077 }
0078
0079
0080
0081 /* Define my thread. */
0082
0083 void    my_thread_entry(ULONG thread_input)
0084 {
0085
0086     UINT    status;
0087     ULONG    actual_status;
0088     NX_PACKET *my_packet;
0089
0090     /* wait for the link to come up. */
0091     do
0092     {
0093
0094         /* Get the link status. */
0095         status = nx_ip_status_check(&my_ip, NX_IP_LINK_ENABLED,
0096             &actual_status, 100);
0097
0098     } while (status != NX_SUCCESS);
0099
0100     /* Create a DHCP instance. */
0101     status = nx_dhcp_create(&my_dhcp, &my_ip, "My DHCP");
0102
0103     /* check for DHCP create error. */
0104     if (status)
0105         error_counter++;

```

```

0106
0107      /* Start DHCP. */
0108      nx_dhcp_start(&my_dhcp);
0109
0110      /* Check for DHCP start error. */
0111      if (status)
0112          error_counter++;
0113
0114      /* Wait for IP address to be resolved through DHCP. */
0115      nx_ip_status_check(&my_ip, NX_IP_ADDRESS_RESOLVED,
0116                        (ULONG *) &status, 100000);
0117
0118      /* Check to see if we have a valid IP address. */
0119      if (status)
0120      {
0121          error_counter++;
0122          return;
0123      }
0124      else
0125      {
0126
0127          /* Yes, a valid IP address is now on lease... All NetX
0128             services are available.
0129          */
0130      }

```

Figure 1.1 Example of DHCP use with NetX

## Multi-Server Environments

---

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

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

## BOOTP Protocol

---

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

## DHCP Multihome Support

---

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

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

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

```

0001 #include "tx_api.h"
0002 #include "nx_api.h"
0003 #include "nx_dhcp.h"
0004
0005 #define DEMO_STACK_SIZE 4096
0006 TX_THREAD my_thread;
0007 NX_PACKET_POOL my_pool;
0008 NX_IP my_ip;
0009 NX_DHCP my_dhcp;
0010
0011 /* Define function prototypes. */
0012
0013 void my_thread_entry(ULONG thread_input);
0014 void my_netx_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
0015
0016 /* Define main entry point. */
0017
0018 intmain()
0019 {
0020
0021     /* Enter the ThreadX kernel. */
0022     tx_kernel_enter();
0023 }
0024
0025 /* Define what the initial system looks like. */
0026
0027 void tx_application_define(void *first_unused_memory)
0028 {
0029     CHAR *pointer;
0030     UINT status;
0031
0032     /* Setup the working pointer. */
0033     pointer = (CHAR *) first_unused_memory;
0034
0035     /* Create "my_thread". */
0036     tx_thread_create(&my_thread, "my thread", my_thread_entry, 0,
0037                     pointer, DEMO_STACK_SIZE,
0038                     2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
0039     pointer = pointer + DEMO_STACK_SIZE;
0040
0041     /* Initialize the NetX system. */
0042     nx_system_initialize();
0043
0044     /* Create a packet pool. */
0045     status = nx_packet_pool_create(&my_pool, "NetX Main Packet Pool",
0046                                   1024, pointer, 64000);
0047     pointer = pointer + 64000;
0048
0049     /* Check for pool creation error. */
0050     if (status)
0051         error_counter++;
0052
0053     /* Create an IP instance without an IP address. */
0054     status = nx_ip_create(&my_ip, "My NetX IP Instance", IP_ADDRESS(0,0,0,0),
0055                           0xFFFFF00, &my_pool, my_netx_driver, pointer, STACK_SIZE, 1);
0056     pointer = pointer + DEMO_STACK_SIZE;
0057
0058     /* Check for IP create errors. */
0059     if (status)

```

```

0063         error_counter++;
0064
0065     status = _nx_ip_interface_attach(&ip_0, "port_2", IP_ADDRESS(0, 0, 0,0),
                                      0xFFFFFFFF00UL, my_netx_driver);

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

```

Figure 1.2 Example of DHCP for NetX with multihome support

## Configuration Options

---



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

Define	Meaning
<b>NX_DHCP_ENABLE_BOOTP</b>	Defined, this option enables the BOOTP protocol instead of DHCP. By default this option is disabled.
<b>NX_DHCP_CLIENT_RESTORE_STATE</b>	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.
<b>NX_DHCP_CLIENT_CLEAR_QUEUE</b>	If defined, this enables the DHCP Client to clear the DHCP Client queue packet. The default value is enabled.
<b>NX_DHCP_CLIENT_USER_CREATE_PACKET_POOL</b>	If set, the DHCP Client will not create its own packet pool. The host application must use the <code>nx_dhcp_packet_pool_set</code> service to set the DHCP Client packet pool. The default value is disabled.
<b>NX_DHCP_CLIENT_SEND_ARP_PROBE</b>	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.
<b>NX_DHCP_CLIENT_SEND_MAX_DHCP_MESSAGE_OPTION</b>	Defined, this enables the DHCP Client to send maximum DHCP message size option. By default, this option is disabled.

<b>NX_DHCP_THREAD_PRIORITY</b>	Priority of the DHCP thread. By default, this value specifies that the DHCP thread runs at priority 3.
<b>NX_DHCP_THREAD_STACK_SIZE</b>	Size of the DHCP thread stack. By default, the size is 2048, which represents a stack of 2048 bytes.
<b>NX_DHCP_TIME_INTERVAL</b>	Number of seconds between iterations of the DHCP client entry thread unaction. By default, this value is 1 second updates.
<b>NX_DHCP_NUM_DNS_SERVERS</b>	Number of DNS name servers the DHCP client will store. By default, this value is 1.
<b>NX_DHCP_NUM_NTP_SERVERS</b>	Number of network time protocol servers the DHCP client will store. By default, this value is 1.
<b>NX_DHCP_PACKET_PAYLOAD</b>	Specifies the size in bytes of the DHCP Client packet payload. The default value is <code>NX_DHCP_MINIMUM_IP_DATAFRAM</code> + physical header size (e.g. Ethernet frame).
<b>NX_DHCP_PACKET_POOL_SIZE</b>	Specifies the size of the DHCP Client packet pool. The default value is $(5 * \text{NX\_DHCP\_PACKET\_PAYLOAD})$ which will provide four packets plus room for internal packet pool overhead.
<b>NX_PACKET_ALLOCATE_TIMEOUT</b>	Specifies the time out option for allocating a packet from the DHCP Client packet pool. The value is defined as the <code>NX_DHCP_IP_PERIODIC_RATE</code> .
<b>NX_DHCP_MIN_RETRANS_TIMEOUT</b>	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\_TIMEOUT\_DECREMENTS**

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

$$(NX\_DHCP\_TIMEOUT\_DECREMENTS * NX\_DHCP\_TIME\_INTERVAL)$$

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

#### **NX\_DHCP\_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.

<b>NX_DHCP_FRAGMENT_OPTION</b>	Fragment enable for DHCP UDP requests. By default, this value is NX_DONT_FRAGMENT to disable DHCP UDP fragmenting.
<b>NX_DHCP_TIME_TO_LIVE</b>	Specifies the number of routers this packet can pass before it is discarded. The default value is set to 0x80.
<b>NX_DHCP_QUEUE_DEPTH</b>	Specifies the number of maximum depth of receive queue. The default value is set to 4.
<b>NX_DHCP_ARP_PROBE_TIMEOUT</b>	Specifies the time out option in timer tick to wait for response to the DHCP Client ARP probe (see NX_DHCP_CLIENT_SEND_ARP_PROBE option). If NX_DHCP_CLIENT_SEND_ARP_PROBE is not enabled, this option has no meaning. The value is defaulted to 10 seconds.

## Chapter 3

# Description of DHCP Client Services

This chapter contains a description of all NetX DHCP services (listed below) in alphabetic order.

In the “Return Values” section in the following API descriptions, values in **BOLD** are not affected by the **NX\_DISABLE\_ERROR\_CHECKING** define that is used to disable API error checking, while non-bold values are completely disabled.

`nx_dhcp_create`  
*Create a DHCP instance*

`nx_dhcp_clear_broadcast_flag`  
Clear broadcast flag on Client messages

`nx_dhcp_delete`  
*Delete a DHCP instance*

`nx_dhcp_decline`  
*Send Decline message to server*

`nx_dhcp_force_renew`  
*Handle Server force renew message*

`nx_dhcp_packet_pool_set`  
*Set the DHCP Client packet pool*

`nx_dhcp_release`  
*Send Release message to server*

`nx_dhcp_reinitialize`  
*Clear DHCP client network parameters*

`nx_dhcp_request_client_ip`  
*Specify a specific IP address*

`nx_dhcp_send_request`  
*Send DHCP message to server*

`nx_dhcp_server_address_get`  
*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*

**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.

**Note:** The DHCP Client packet pool payload is defaulted to NX\_DHCP\_PACKET\_PAYLOAD (548 bytes plus UDP, IP and Ethernet headers). 548 bytes is the mandatory payload size a DHCP Client should be able to receive.

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to DHCP control block.
<b>ip_ptr</b>	Pointer to previously created IP instance.
<b>name_ptr</b>	Pointer to name for DHCP instance.

**Return Values**

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

**Allowed From**

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. */
```

**See Also**

nx\_dhcp\_delete, nx\_dhcp\_request\_client\_ip,  
nx\_dhcp\_set\_interface\_index, nx\_dhcp\_release, nx\_dhcp\_start,  
nx\_dhcp\_state\_change\_notify, nx\_dhcp\_stop

**nx\_dhcp\_clear\_broadcast\_flag**

Send DHCP messages with the broadcast flag cleared

**Prototype**

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

**Description**

This service enables the DHCP Client host application to have the broadcast flag cleared in DHCP Client messages to the DHCP Server. On certain message types (DISCOVER) the DHCP Client will set the broadcast flag to request the Server reply with a broadcast. This is typically because the Client does not have an IP address. If `clear_flag` is set to `NX_TRUE`, the broadcast flag is cleared where it would normally set the bit on. If it is `NX_FALSE`, the DHCP Client will set the broadcast flag (default behavior)

It is intended for DHCP Clients going through a router to the DHCP Server, where the router rejects the message because of the broadcast request.

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to DHCP control block
<b>clear_flag</b>	Value to set the broadcast flag to

**Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful DHCP create
<b>NX_PTR_ERROR</b>	(0x16)	Invalid IP or DHCP pointer
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service
<b>NX_NOT_ENABLED</b>	(0x14)	UDP not enabled on IP instance

**Allowed From**

Threads

**Example**

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

/* If status is NX_SUCCESS the DHCP Client messages will request unicast replies.
   */
```



**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**

<b>dhcp_ptr</b>	Pointer to previously created DHCP instance.
-----------------	--

**Return Values**

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

**Allowed From**

Threads

**Example**

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

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

**See Also**

nx\_dhcp\_create, nx\_dhcp\_release, nx\_dhcp\_start,  
nx\_dhcp\_state\_change\_notify, nx\_dhcp\_stop

**nx\_dhcp\_decline**

Decline a Leased IP address

**Prototype**

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

**Description**

This service informs the DHCP Server the DHCP Client is declining the IP address offered by the DHCP Server and returns the DHCP state machine to the initial state. A new IP address can be requested by calling *nx\_dhcp\_start* again. The host application must use this service and not *nx\_dhcp\_send\_request* to send a DECLINE message.

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to previously created DHCP instance.
-----------------	--

**Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful DHCP release
<b>NX_DHCP_NOT_STARTED</b>	(0x96)	DHCP Client not started
<b>NX_PTR_ERROR</b>	(0x16)	Invalid DHCP pointer
<b>NX_CALLER_ERROR</b>	(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 DECLINE message was successfully sent. */
```

**See Also**

*nx\_dhcp\_create*, *nx\_dhcp\_decline*, *nx\_dhcp\_delete*, *nx\_dhcp\_start*,  
*nx\_dhcp\_state\_change\_notify*, *nx\_dhcp\_stop*

**nx\_dhcp\_force\_renew**

Handle a server force renew message

**Prototype**

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

**Description**

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

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to previously created DHCP instance.
-----------------	--

**Return Values**

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

**Allowed From**

Threads

**Example**

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

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

**See Also**

nx\_dhcp\_create, nx\_dhcp\_delete, nx\_dhcp\_start,  
nx\_dhcp\_state\_change\_notify, nx\_dhcp\_stop

**nx\_dhcp\_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 sets the DHCP Client packet pool by passing in a pointer to a previously created packet pool. To use this service, the host application must define `NX_DHCP_CLIENT_USER_CREATE_PACKET_POOL` so that the *nx\_dhcp\_create* service will not create the Client's packet pool. Note that the caller should 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**

<b>dhcp_ptr</b>	Pointer to DHCP control block.
<b>packet_pool_ptr</b>	Pointer to previously created packet pool

**Return Values**

<b>NX_SUCCESS</b>	(0x00)	DHCP Client packet pool is set
<b>NX_PTR_ERROR</b>	(0x16)	Invalid DHCP pointer

**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. */
```

**See Also**

`nx_dhcp_delete`, `nx_dhcp_create`, `nx_dhcp_release`, `nx_dhcp_start`,  
`nx_dhcp_state_change_notify`, `nx_dhcp_stop`

**nx\_dhcp\_reinitialize**

Clear the DHCP client network parameters

**Prototype**

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

**Description**

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

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

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to previously created DHCP instance.
-----------------	--

**Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful DHCP release
<b>NX_PTR_ERROR</b>	(0x16)	Invalid DHCP pointer

**Allowed From**

Threads

**Example**

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

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

**See Also**

*nx\_dhcp\_create*, *nx\_dhcp\_delete*, *nx\_dhcp\_start*,  
*nx\_dhcp\_state\_change\_notify*, *nx\_dhcp\_stop*

**nx\_dhcp\_release**

Release Leased IP address

**Prototype**

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

**Description**

This service releases the IP address obtained from the previous DHCP start request and returns the DHCP state machine to the initial state. The host application must use this service and not *nx\_dhcp\_send\_request* to send a RELEASE message. A new IP address can be requested by calling *nx\_dhcp\_start* again.

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to previously created DHCP instance.
-----------------	--

**Return Values**

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

**Allowed From**

Threads

**Example**

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

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

**See Also**

*nx\_dhcp\_create*, *nx\_dhcp\_decline*, *nx\_dhcp\_delete*, *nx\_dhcp\_start*, *nx\_dhcp\_state\_change\_notify*, *nx\_dhcp\_stop*

**nx\_dhcp\_request\_client\_ip**

Set requested IP address for DHCP instance

**Prototype**

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

**Description**

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

**Input Parameters**

**dhcp\_ptr** Pointer to DHCP control block.  
**client\_ip\_address** IP address to request from DHCP server  
**skip\_discover\_message** If true, DHCP Client sends Request message; else it starts with the Discover message.

**Return Values**

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

**Allowed From  
Threads****Example**

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

**See Also**

nx\_dhcp\_delete, nx\_dhcp\_create, nx\_dhcp\_release, nx\_dhcp\_start,  
 nx\_dhcp\_state\_change\_notify, nx\_dhcp\_stop

**nx\_dhcp\_send\_request**

Send DHCP message to Server

**Prototype**

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

**Description**

This service sends a message to the DHCP server. This is intended primarily for the host application to send INFORM REQUEST messages. It is NOT intended for the host application to drive the DHCP Client state machine. It does not update the DHCP client internally. If the host needs to decline or release an IP address leased from a DHCP Server it should use the *nx\_dhcp\_decline* or *nx\_dhcp\_releaseservice* respectively.

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to DHCP control block.
<b>dhcp_message_type</b>	Message request (defined in <i>nx_dhcp.h</i> )

**Return Values**

<b>NX_DHCP_INVALID_MESSAGE</b>	(0x9B)	Illegal message type
<b>NX_DHCP_NOT_STARTED</b>	(0x96)	Invalid interface index
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input

**Allowed From**

Threads

**Example**

```
/* Send the INFORM REQUEST message to the DHCP server. It is assumed the DHCP
Client task is already started */

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

**See Also**

*nx\_dhcp\_release*, *nx\_dhcp\_decline*, *nx\_dhcp\_state\_change\_notify*,  
*nx\_dhcp\_reinitialize*, *nx\_dhcp\_stop*, *nx\_dhcp\_start*



**nx\_dhcp\_server\_address\_gset**

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. The caller should use this service when the DHCP Client has been granted an IP address and is in a bound state. The host application can either use the *nx\_ip\_status\_check* service or a successful ping exchange to verify IP address is set, or can use the *nx\_dhcp\_state\_change\_notify* and query the DHCP Client state is NX\_DHCP\_STATE\_BOUND.

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to DHCP control block.
<b>server_address</b>	Pointer to server IP address

**Return Values**

<b>NX_SUCCESS</b>	(0x00)	DHCP server address returned
<b>NX_PTR_ERROR</b>	(0x16)	Invalid input pointer

**Allowed From**

Application code

**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);
    }
}*/
```

**See Also**

*nx\_dhcp\_delete*, *nx\_dhcp\_create*, *nx\_dhcp\_release*, *nx\_dhcp\_start*,  
*nx\_dhcp\_state\_change\_notify*, *nx\_dhcp\_stop*

**nx\_dhcp\_set\_interface\_index**

Set network interface for DHCP instance

**Prototype**

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

**Description**

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

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

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to DHCP control block.
<b>index</b>	Index of device network interface

**Return Values**

<b>NX_SUCCESS</b>	(0x00)	Interface is successfully set.
<b>NX_DHCP_BAD_INTERFACE_INDEX_ERROR</b>	(0x9A)	Invalid interface index
<b>NX_PTR_ERROR</b>	(0x16)	Invalid DHCP pointer

**Allowed From**  
Threads

**Example**

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

**See Also**

nx\_dhcp\_delete, nx\_dhcp\_request\_client\_ip, nx\_dhcp\_create,  
nx\_dhcp\_release, nx\_dhcp\_start, nx\_dhcp\_state\_change\_notify,  
nx\_dhcp\_stop

**nx\_dhcp\_start**

Start DHCP processing

**Prototype**

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

**Description**

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

Note that when proceeding further, the application should use *nx\_ip\_status\_check* to see when an IP address is obtained.

**Input Parameters**

**dhcp\_ptr**                      Pointer to previously created DHCP instance.

**Return Values**

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

**Allowed From**

Threads

**Example**

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

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

**See Also**

*nx\_dhcp\_create*, *nx\_dhcp\_delete*, *nx\_dhcp\_release*,  
*nx\_dhcp\_state\_change\_notify*, *nx\_dhcp\_stop*, *nx\_dhcp\_request\_client\_ip*,  
*nx\_dhcp\_set\_interface\_index*

**nx\_dhcp\_state\_change\_notify**

Notify application of DHCP state change

**Prototype**

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

**Description**

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

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

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to previously created DHCP instance.
<b>dhcp_state_change_notify</b>	Application callback function pointer

**Return Values**

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

**Allowed From**

Threads

**Example**

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

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

### See Also

`nx_dhcp_create`, `nx_dhcp_start`, `nx_dhcp_stop`,  
`nx_dhcp_user_option_retrieve`, `nx_dhcp_user_option_convert`

**nx\_dhcp\_stop**

Stops DHCP processing

**Prototype**

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

**Description**

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

**Input Parameters**

**dhcp\_ptr**                      Pointer to previously created DHCP instance.

**Return Values**

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

**Allowed From**

Threads

**Example**

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

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

**See Also**

nx\_dhcp\_create, nx\_dhcp\_delete, nx\_dhcp\_release, nx\_dhcp\_start,  
nx\_dhcp\_state\_change\_notify

**nx\_dhcp\_user\_option\_retrieve**

Retrieve a DHCP option from last server response

**Prototype**

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

**Description**

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

**Input Parameters**

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

## Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful DHCP option retrieval.
<b>NX_DHCP_NOT_BOUND</b>	(0x94)	The IP address has not been leased yet so option requests cannot be made.
<b>NX_DHCP_ERROR</b>	(0x90)	Option not found in buffer. Please include the option in the <code>_nx_dhcp_request_parameters</code> which is defined at the top of <i>nx_dhcp.c</i> .
<b>NX_DHCP_DEST_TO_SMALL</b>	(0x95)	Destination is too small to hold response.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid DHCP or destination pointer.
<b>NX_CALLER_ERROR</b>	(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(dns_ip_string);
status = nx_dhcp_user_option_retrieve(&my_dhcp, NX_DHCP_OPTION_DNS_SVR,
                                     dns_ip_string, &size);

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

```

## See Also

`nx_dhcp_user_option_convert`



**nx\_dhcp\_user\_option\_convert**

Convert four bytes to ULONG

**Prototype**

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

**Description**

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

**Input Parameters**

<b>option_string_ptr</b>	Pointer to previously retrieved option string.
--------------------------	--

**Return Values**

<b>Value</b>	Value of first four bytes.
--------------	----------------------------

**Allowed From**

Threads

**Example**

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

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

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

**See Also**

nx\_dhcp\_stop, nx\_dhcp\_user\_option\_retrieve

# Appendix A - Description of the Restore State Feature

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

This option also allows an application to suspend the DHCP Client thread and resume it, updated with the elapsed time between suspending and resuming the thread without powering down.

## Restoring the DHCP Client between Reboots

---

To restore a DHCP Client between reboots, the DHCP application creates an instance of the DHCP Client, and then obtains an IP address lease using the normal DHCP protocol and calling `nx_dhcp_start`. Then the DHCP application waits for the protocol to complete. If all goes well, the device achieves the BOUND state with an assigned valid IP address from its DHCP Server. Before it powers down, the DHCP application saves the current DHCP Client instance to a DHCP Client record which is then stored in non-volatile memory. An independent 'time keeper' elsewhere in the system keeps 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. 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.

```
/* 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;
    while(1)
    {
        /* Wait for DHCP to assign the 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(...);
    }
}

```

**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 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**

<b>dhcp_ptr</b>	Pointer to DHCP Client
<b>record_ptr</b>	Pointer to DHCP Client record

**Return Values**

<b>NX_SUCCESS</b>	(0x0)	Valid Client record created
<b>NX_DHCP_NOT_BOUND</b>	(0x94)	Client not in bound state, therefore not assigned valid IP address
<b>NX_PTR_ERROR</b>	(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. */
```

**See Also**

nx\_dhcp\_resume, nx\_dhcp\_suspend, nx\_dhcp\_client\_restore\_record,  
nx\_dhcp\_client\_update\_time\_remaining

**nx\_dhcp\_client\_restore\_record**

Restore DHCP Client state from 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 a DHCP application to recreate its DHCP Client state from a previous session by updating the DHCP Client with the DHCP Client record pointed to by `record_ptr`, and updates the time remaining on DHCP Client lease with the `time_elapsed` input. This allows the DHCP Client application to recreate its DHCP Client, for example, after powering down. This requires that the DHCP Client application created a record of the DHCP Client before powering down, and saved that record to nonvolatile memory.

**Input Parameters**

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

**Return Values**

<b>NX_SUCCESS</b>	(0x0)	Client record restored
<b>status</b>		Status completion from native NetX library calls
<b>NX_PTR_ERROR</b>	(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. */
```

**See Also**

`nx_dhcp_resume`, `nx_dhcp_suspend`, `nx_dhcp_client_get_record`,  
`nx_dhcp_client_update_time_remaining`

**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. The DHCP Client must suspend the DHCP Client thread before using this service using `nx_dhcp_suspend`. After calling this service, the application then resumes 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**

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

**Return Values**

<b>NX_SUCCESS</b>	(0x0)	Client IP lease updated
<b>NX_PTR_ERROR</b>	(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. */
```

### See Also

`nx_dhcp_resume`, `nx_dhcp_suspend`, `nx_dhcp_client_get_record`,  
`nx_dhcp_client_restore_record`

**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.

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**

<b>dhcp_ptr</b>	Pointer to DHCP Client
-----------------	------------------------

**Return Values**

<b>NX_SUCCESS</b>	(0x0)	Client thread is suspended
<b>NX_PTR_ERROR</b>	(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. */
```

**See Also**

*nx\_dhcp\_resume*, *nx\_dhcp\_client\_update\_remaining\_time*,  
*nx\_dhcp\_client\_get\_record*, *nx\_dhcp\_client\_restore\_record*

**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.

**Input Parameters**

<b>dhcp_ptr</b>	Pointer to DHCP Client
-----------------	------------------------

**Return Values**

<b>NX_SUCCESS</b>	(0x0)	Client thread is resumed
<b>NX_PTR_ERROR</b>	(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. */
```

**See Also**

nx\_dhcp\_suspend, nx\_dhcp\_client\_update\_remaining\_time,  
nx\_dhcp\_client\_get\_record, nx\_dhcp\_client\_restore\_record