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

Dynamic Host Configuration Protocol for Servers (NetX DHCP Server)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1050
Revision 5.0

Contents

Chapter 1

Introduction to DHCP Server

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 application establishes communication with its DHCP server to dynamically request and 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

The DHCP Server utilizes the UDP protocol to receive DHCP Client requests and transmit 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. However, if the Client knows the address of the DHCP Server it may send DHCP messages using unicast messages.

DHCP Server State Machine

The DHCP Server is implemented as a two step state machine processed by an internal DHCP thread that is created during *nx_dhcp_server _create* processing. The main states of DHCP Server are 1) receiving a DISCOVER message from a DHCP client and 2) receiving a REQUEST message.

Below are the corresponding DHCP Client states:

State	**Meaning**

NX_DHCP_STATE_BOOT	Starting with a previous IP address
NX_DHCP_STATE_INIT	Starting with no previous IP address value
NX_DHCP_STATE_SELECTING	Waiting for a response from any DHCP server
NX_DHCP_STATE_REQUESTING	DHCP Server identified, IP address request sent
NX_DHCP_STATE_BOUND	DHCP IP Address lease established
NX_DHCP_STATE_RENEWING	DHCP IP Address lease renewal time elapsed, renewal requested
NX_DHCP_STATE_REBINDING	DHCP IP Address lease rebind time elapsed, renewal requested
NX_DHCP_STATE_FORCERENEW	DHCP IP Address lease established, force renewal by server or by application
NX_DHCP_STATE_FAILED	No server found or no response from server received

DHCP Additional Parameters

The NetX DHCP Server has a default list of option parameters which is set in the configurable option NX_DHCP_DEFAULT_SERVER_OPTION_LIST in *nx_dhcp_server.h* to supply DHCP Clients with common/critical network configuration parameters e.g. router or gateway address and DNS server for the DHCP Client.

DHCP RFCs

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

Chapter 2

Installation and Use of the NetX DHCP Server

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

Product Distribution

The NetX DHCP Server 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_server.h	Header file for NetX DHCP Server
nx_dhcp_server.c	C Source file for NetX DHCP Server
nx_dhcp_server.pdf	PDF description of NetX DHCP Server
demo_netx_dhcp_server.c	NetX DHCP Server demonstration

DHCP Installation

In order to use NetX DHCP Server, 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_server.h* and *nx_dhpc_server.c* files should be copied into this directory.

Using NetX DHCP Server

Using NetX DHCP Server is easy. Basically, the application code must include *nx_dhcp_server.h* after it includes *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX, respectively. Once *nx_dhcp_server.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_server.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. For more details on using NetX DHCP Server, see the following sections **Requirements of the NetX DHCP Server** and **Constraints of the NetX DHCP Server**.

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

Requirements of the NetX DHCP Server

The NetX DHCP Server requires a UDP socket port assigned to the well known DHCP port 67. To create the DHCP Server, the application must create a packet pool with packet payload at least 548 bytes plus IP, UDP and Ethernet headers (which total 44 bytes with 4 byte alignment).

It is assumed that the Server and Client are both using Ethernet hardware address settings:

Hardware type	1
Hardware length	6
Hops	0

Multiple Client Sessions

The NetX DHCP Server can handles multiple Client sessions by keeping a table of active DHCP clients and what 'state' the Client is in e.g. DHCP states INIT, BOOT, SELECTING, REQUESTING, RENEWING etc. If the session time out expires before receiving the next Client message, unless that Client is bound to an IP lease, the Server will clear the Client session data and return the assigned IP address back to the available pool. If the Server receives multiple DISCOVER messages from the same Client the Server resets the session time out and keeps the IP address reserved for the Client to accept in a subsequent REQUEST message.

The NetX DHCP Server also accepts the single state Client DHCP request e.g. the Client only sends a REQUEST message. This assumes the Client has been previously assigned an IP lease from the DHCP server.

The NetX DHCP Server uses DECLINE, RELEASE and INFORM messages to update IP address availability status in its IP address database.

Setting Interface Specific Network Parameters Server Responses

The application can set the router, subnet mask and DNS server parameters for each interface it handles DHCP Client requests, using the *nx_dhcp_set_interface_network_parameters* service. Otherwise these parameters are defaulted to the IP gateway on the Server's primary interface, its DHCP network subnet, and DHCP Server IP address, respectively.

The DHCP server includes these parameters in the option data of DHCP messages it sends to DHCP clients.

Assigning IP addresses to the Client

If the Client DISCOVER message does not specify a requested IP address, the DHCP Server can use one from its own pool. If the Server has no available IP addresses it will send the Client a NACK message.

The NetX DHCP Server will grant the requested IP address in the Client REQUEST message as long as the IP address is available and can be found in the Server IP address database. The application creates the Server's list of available IP addresses for assigning to DHCP Clients using the *nx_dhcp_create_server_ip_address_list* service. If the Server does not have the requested IP addresses or it is assigned to another host it will send the Client a NACK message.

When the DHCP Server receives a Client request, it identifies that Client uniquely using the Client MAC address in the Client MAC address field in the DHCP message. If the Client changes it's MAC address or is moved elsewhere onto another subnet it should send a RELEASE message to the Server to return the IP address back to the available pool, and request a new IP address in the INIT state.

See Figure 1.1 of the **Small Example System** section for details. The number of IP addresses saved to the DHCP Server instance is limited to the size of the server address array in the DHCP Server control block, and defined by the configurable option NX_DHCP_IP_ADDRESS_LIST_SIZE.

IP Address Lease Times

The DHCP Server will also accept the request Client lease time if that lease time is less than the Server default lease time which is defined in configurable option NX_DHCP_ASSIGNED_LEASE_TIME. Renewal and rebind times assigned to the Client are 50% and 85% of the lease time, respectively, unless the lease time is infinity (0xFFFFFFFF), in which case renewal and rebind times are also set to infinity.

DHCP Server Timeouts

The DHCP Server has a user configurable session timeout, NX_DHCP_CLIENT_SESSION_TIMEOUT, for waiting for the next DHCP Client message unless the session is completed. The time out is reset when the Server receives the next message from the Client regardless if is the same message previously sent.

Internal error handling

The DHCP Server receives and processes DHCP Client packets in the *nx_dhcp_listen_for_messages* function. This function will discontinue processing the current DHCP Client packet if the packet is invalid, or the DHCP Server encounters an internal error. n*x_dhcp_listen_for_messages* returns an error status. The DHCP Server thread relinquishes control briefly of the ThreadX scheduler before calling this function to receive the next DHCP Client message. In the current release there is no logging support for error status returns from *nx_dhcp_listen_for_messages.*

Option 55: Parameter Request List

The NetX DHCP Server must be configured with a set of options to load to Parameter Request Option (55) list in the OFFER and DHCPACK messages it transmits back to the Client. These options should include network critical configuration data for the Client network and by default is defined to be router IP address, subnet mask, and DNS server. The option list is a space delimited list and defined in the user configurable NX_DHCP_DEFAULT_SERVER_OPTION_LIST. Note the number of options specified in this list must equal NX_DHCP_DEFAULT_OPTION_LIST_SIZE which is also user defined.

Constraints of the NetX DHCP Server

DHCP Messages

The NetX DHCP Server does not verify that an IP address has not been assigned elsewhere on the network before granting the IP address to the Client. If there are multiple DHCP servers, this can indeed be the case. *As per RFC 2131, it is the Client's responsibility to verify the IP address is unique on its network* (e.g. pinging the address). If it is not, the Server should receive a DECLINE message with the IP address to update its database from the Client.

The NetX DHCP Server does not issue FORCE_RENEW messages. It is up to the DHCP Client to renew its IP address lease. However, the DHCP Server monitors the time remaining on all the assigned IP addresses in its database. When an IP address lease expires that IP address is returned to the pool of available IP addresses. Hence it is up to the Client to actively renew/rebind its IP address lease.

Session data is cleared as soon as the Client either is granted (“bound”) to an IP lease (or an existing one is renewed). If a Client packet proves bogus, or the Client times out between responses, session data is cleared.

Saving Data Between Reboots

The NetX DHCP Server saves Client data including DHCP request parameters in a Client record table. This table is not stored in non volatile memory, so if the DHCP Server host must reboot that information is not saved between reboots.

The NetX DHCP Server saves IP address lease data in a IP address table. This table is not stored in non volatile memory, so if the DHCP Server host must reboot that information is not saved between reboots.

Relay Agents

The NetX DHCP Server is configured with a zero IP address for the ‘Relay agent’ field because it does not support out of network DHCP requests.

Small Example System

An example of how easy it is to use the NetX DHCP Server is described in Figure 1.1 that appears below. In this example, the DHCP include file *nx_dhcp_server.h* is brought in at line 5. DHCP Server thread stack size, IP thread stack size and test thread stack size are all defined in lines 7-13.

First, an optional test thread task for stopping, restarting and eventually deleting the DHCP server is created with the "*test_thread_entry*" function at line 57. A DHCP Server control block "*dhcp_server*" is defined as a global variable at line 20. Note that the server packet pool is created with packets having a payload at least as large as the standard DHCP message (548 bytes plus IP and UDP header bytes). After successfully creating an IP instance for the DHCP Server, the application creates the DHCP Server in line 96. Next, the application enables the Server IP instance to be UDP enabled. Before starting the DHCP Server, the available IP address list is created in line 137 using the **nx_dhcp_create_server_ip_address_list** service. The network configuration parameters are set in the following line 138 using the **nx_dhcp_set_interface_network_parameters** service, DHCP Server services become available when the application calls the *nx_dhcp_server_start* at line 141. The test thread task demonstrates the use of stopping and restarting the DHCP server.

```
1   /* This is a small demo of NetX DHCP Server for the high-performance NetX TCP/IP stack.  */
2
3   #include   "tx_api.h"
4   #include   "nx_api.h"
5   #include   "nx_dhcp_server.h"
6
7   #define     DEMO_TEST_STACK_SIZE         2048
8   #define     DEMO_SERVER_STACK_SIZE  2048
9   #define     SERVER_IP_ADDRESS_LIST  "192.168.2.10 192.168.2.11 192.168.2.12"
10  #define     PACKET_PAYLOAD          1000
11  #define     PACKET_POOL_SIZE        (PACKET_PAYLOAD * 10)
12  #define     SERVER_IP_THREAD_STACK    2048
13
14
15  /* Define the ThreadX and NetX object control blocks...  */
16
17  TX_THREAD                       test_thread;
18  NX_PACKET_POOL                  server_pool;
19  NX_IP                           server_ip;
20  NX_DHCP_SERVER                  dhcp_server;
21
22
23  /* Define the counters used in the demo application...  */
24
25  ULONG                            state_changes;
26
27
28  /* Define thread prototypes.  */
29
30  void    test_thread_entry(ULONG thread_input);
31  void    nx_etherDriver_mcf5485(struct NX_IP_DRIVER_STRUCT *driver_req);
```

```
32
33
34  /* Define main entry point.  */
35
36  int main()
37  {
38
39      /* Enter the ThreadX kernel.  */
40      tx_kernel_enter();
41  }
42
43
44  /* Define what the initial system looks like.  */
45
46  void    tx_application_define(void *first_unused_memory)
47  {
48
49  CHAR    *pointer;
50  UINT    status;
51
52
53      /* Setup the working pointer.  */
54      pointer =  (CHAR *) first_unused_memory;
55
56      /* Create the test thread.  */
57      status = tx_thread_create(&test_thread, "test thread", test_thread_entry, 0,
58            pointer, TEST_STACK_SIZE,  1, 1, TX_NO_TIME_SLICE, TX_DONT_START);
59
60      if (status)
61      {
62          printf("Error with DHCP test thread create. Status 0x%x\r\n", status);
63          return;
64      }
65
66      pointer =  pointer + DEMO_STACK_SIZE;
67
68      /* Initialize the NetX system.  */
69      nx_system_initialize();
70
71      /* Create the DHCP Server packet pool.  */
72      status =  nx_packet_pool_create(&server_pool, "NetX Main Packet Pool", PACKET_PAYLOAD,
                                  pointer, PACKET_POOL_SIZE);
73      pointer = pointer + PACKET_POOL_SIZE;
74
75      /* Check for pool creation error.  */
76      if (status)
77      {
78          printf("Error with DHCP server packet pool create. Status 0x%x\r\n", status);
79          return;
80      }
81
82      /* Create the DHCP Server IP instance.  */
83      status = nx_ip_create(&server_ip, "NetX DHCP Server IP", NX_DHCP_SERVER_IP_ADDRESS,
                                  0xFFFFFF00UL,  &server_pool, nx_etherDriver_mcf5485, pointer,
                                  SERVER_IP_THREAD_STACK, 1);
85
86      pointer =  pointer + DEMO_IP_THREAD_STACK;
87
88      /* Check for IP create errors.  */
89      if (status)
90      {
91          printf("Error with DHCP server IP task create. Status 0x%x\r\n", status);
```

```
92          return;
93      }
94
95      /* Create the DHCP Server instance.  */
96      status =  nx_dhcp_server_create(&dhcp_server, &server_ip, pointer,
97                                      DEMO_SERVER_STACK_SIZE,"DHCP Server", &server_pool);
98
99      if (status)
100     {
101         printf("Error with DHCP server create. Status 0x%x\r\n", status);
102         return;
103     }
104
105     pointer = pointer + DEMO_SERVER_STACK_SIZE;
106
107     /* Enable ARP and supply ARP cache memory for IP Instance 0.  */
108     status =  nx_arp_enable(&server_ip, (void *) pointer, 1024);
109     pointer = pointer + 1024;
110
111     /* Check for ARP enable errors.  */
112     if (status)
113     {
114         printf("Error with ARP enable. Status 0x%x\r\n", status);
115         return;
116     }
117
118     /* Enable UDP traffic.  */
119     status =  nx_udp_enable(&server_ip);
120
121     /* Check for UDP enable errors.  */
122     if (status)
123     {
124         printf("Error with ICMP enable. Status 0x%x\r\n", status);
125         return;
126     }
127
128     /* Enable ICMP to enable the ping utility.  */
129     status =  nx_icmp_enable(&server_ip);
130
131     /* Check for errors.  */
132     if (status)
133     {
134         printf("Error with ICMP enable. Status 0x%x\r\n", status);
135     }
136
137     status = nx_dhcp_create_server_ip_address_list(&dhcp_server, iface_index,
                                START_IP_ADDRESS_LIST, END_IP_ADDRESS_LIST, &addresses_added);

138     status = nx_dhcp_set_interface_network_parameters(&dhcp_server, iface_index,
                                NX_DHCP_SUBNET_MASK,    NX_DHCP_DEFAULT_GATEWAY,
                                NX_DHCP_DNS_SERVER);
139
140     /* Start the DHCP Server.  */
141     status =  nx_dhcp_server_start(&dhcp_server);
142
143     tx_thread_resume(&test_thread);
144 }
145
146 /* Define the test thread.  */
147 void    test_thread_entry(ULONG thread_input)
148 {
149
```

```
150 UINT       status;
151 UINT       keep_spinning;
152
153
154     /* Just let the test thread be idle till we're ready to shut things down. */
155     keep_spinning = 1;
156     while(keep_spinning)
157     {
158         tx_thread_sleep(300);
159     }
160
161     printf("Stopping the server...\n");
162     status = nx_dhcp_server_stop(&dhcp_server);
163     if (status)
164     {
165         printf("Error with DHPC server stop. Status 0x%x\r\n", status);
166         return;
167     }
168
169     tx_thread_sleep(500);
170
171     printf("Starting the server...\n");
172     status = nx_dhcp_server_start(&dhcp_server);
173     if (status)
174     {
175         printf("Error with DHPC server start. Status 0x%x\r\n", status);
176         return;
177     }
178
179
180     tx_thread_sleep(600);
181
182     printf("Stopping the server for good...\n");
183     status = nx_dhcp_server_stop(&dhcp_server);
184     if (status)
185     {
186         printf("Error with DHPC server stop. Status 0x%x\r\n", status);
187         return;
188     }
189
190     tx_thread_sleep(200);
191
192
193     printf("Deleting the server...\n");
194     status = nx_dhcp_server_delete(&dhcp_server);
195     if (status)
196     {
197         printf("Error with DHCP server delete. Status 0x%x\r\n", status);
198         return;
199     }
200 }
```

Figure 1.1 Example NetX DHCP Server application

Configuration Options

There are several configuration options for building NetX DHCP Server. The following list describes each in detail:

Define	Meaning
NX_DISABLE_ERROR_CHECKING	This option removes the basic DHCP error checking. It it typically used after the application is debugged.
NX_DHCP_SERVER_THREAD_PRIORITY	This option specifies the priority of the DHCP Server thread. By default, this value specifies that the DHCP thread runs at priority 1.
NX_DHCP_TYPE_OF_SERVICE	This option specifies the 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 set to NX_DONT_FRAGMENT to disable UDP fragmenting.
NX_DHCP_TIME_TO_LIVE	Specifies the number of routers the packet can pass before it is discarded. The default value is 0x80.
NX_DHCP_QUEUE_DEPTH	Specifies the number of packets that the DHCP Server socket keeps before flushing the queue. The default value is 5.
NX_DHCP_PACKET_ALLOCATE_TIMEOUT	Specifies the timeout in timer ticks for the NetX DHPC Server to wait to allocate a packet from its packet pool. The default value is 200.

NX_DHCP_SERVER_IP_ADDRESS This is the DHCP Server IP address for the client host subnet.

NX_DHCP_SERVER_ID This is the DHCP Server ID which the Client uses to communicate which DHCP Server it is choosing. The default value is set to NX_DHCP_SERVER_IP_ADDRESS.

NX_DHCP_ROUTER_IP_ADDRESS The router IP address for the client host subnet. The default value is set to NX_DHCP_SERVER_IP_ADDRESS.

NX_DHCP_DNS_IP_ADDRESS The DNS IP address for the client host subnet. The default value is set to NX_DHCP_SERVER_IP_ADDRESS.

NX_DHCP_SUBNET_MASK This is the subnet mask the DHCP Client should be configured with. The default value is set to 0xFFFFFF00.

NX_DHCP_CLIENT_IDENTIFIER_MAX

This sets the maximum limit on the length of the Client Identifier string, which is defaulted to the Client MAC address (6 bytes)

NX_DHCP_FAST_PERIODIC_TIME_INTERVAL

This is timeout period in timer ticks for the DHCP Server fast timer to check on session time remaining and handle sessions that have timed out.

NX_DHCP_CLIENT_SESSION_TIMEOUT

This is timeout period in timer ticks the DHCP Server will wait to receive the next DHCP Client message.

NX_DHCP_ASSIGNED_LEASE_TIME This is IP Address lease time in seconds assigned to the DHCP Client, and the basis for computing the renewal and rebind times also

assigned to the Client. The default value is set to 0xFFFFFFFF (infinity).

NX_DHCP_IP_ADDRESS_LIST_SIZE This is size of the DHCP Server array for holding available IP addresses for assigning to the Client. Note that the actual IP list is not a configurable option but passed as a parameter in the DHCP server create call. The default value is 50.

NX_DHCP_CLIENT_OPTIONS_MAX This is size of the array in the DHCP Client instance for holding the all the requested options in the parameter request list in the current session. The default value is 12.

NX_DHCP_DEFAULT_SERVER_OPTION_LIST

This is the buffer holding the DHCP Server's default list of options to supply to the current DHCP Client in the parameter request list. The default is "1 3 6."

NX_DHCP_DEFAULT_SERVER_OPTION_LIST

This is the size of the array to hold the DHCP Server's default list of options. The default value is 3.

NX_DHCP_CLIENT_HOSTNAME_MAX

This is size of the buffer for holding the Client host name in the current DHCP Server Client session. The default value is 30.

Chapter 3

Description of DHCP Server Services

This chapter contains a description of all NetX DHCP Server 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_server_create
Create a DHCP Server instance

nx_dhcp_set_interface_network_parameters
Set DHCP Server options for critical network parameters for specified interface

nx_dhcp_create_server_ip_address_list
Create pool of available IP addresses to assign to DHCP Clients interface

nx_dhcp_clear_client_record
Remove Client record in the Server database

nx_dhcp_server_delete
Delete a DHCP Server instance

nx_dhcp_server_start
Start or resume DHCP Server processing

nx_dhcp_server_stop
Stop DHCP server processing

nx_dhcp_server create

Create a DHCP Server instance

Prototype

```
UINT nx_dhcp_server_create(NX_DHCP_SERVER *dhcp_ptr, NX_IP *ip_ptr,
                           VOID *stack_ptr, ULONG stack_size,
                           CHAR *input_address_list, CHAR *name_ptr,
                           NX_PACKET_POOL *packet_pool_ptr);
```

Description

This service creates a DHCP Server instance with a previously created IP instance.

Important Note: The application must make sure the packet pool created for the IP create service has a minimum 548 byte payload, not including the UDP, IP and Ethernet headers.

Input Parameters

dhcp_ptr	Pointer to DHCP Server control block.
ip_ptr	Pointer to DHCP Server IP instance.
stack_ptr	Pointer DHCP Server stack location.
stack_size	Size of DHCP Server stack
input_address_list	Pointer to Server's list of IP addresses
name_ptr	Pointer to DHCP Server name
packet_pool_ptr	Pointer to DHCP Server packet pool

Return Values

NX_SUCCESS	(0x00)	Successful DHCP Server create.
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_DHCP_INADEQUATE_PACKET_POOL_PAYLOAD	(0xA9)	Packet payload too small error
NX_DHCP_BAD_SERVER_IP_CONFIGURATION	(0xA5)	Bad Server IP address error
NX_DHCP_NO_DEFAULT_OPTION_LIST	(0xA4)	Missing option list error
status		Completion status of socket and thread create calls

Allowed From

Application

Example

```
/* Create a DHCP Server instance.  */
status =  nx_dhcp_server_create(&dhcp_server, &server_ip, pointer,
                                DEMO_SERVER_STACK_SIZE, SERVER_IP_ADDRESS_LIST, "DHCP
                                 server", &server_pool);

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

nx_dhcp_create_server_ip_address_list

Create a IP address pool

Prototype

```
UINT nx_dhcp_create_server_ip_address_list(NX_DHCP_SERVER *dhcp_ptr,
                        UINT iface_index, ULONG start_ip_address,
                        ULONG end_ip_address, UINT *addresses_added);
```

Description

This service creates a network interface specific pool of available IP addresses for the specified DHCP server to assign. The start and end IP addresses must match the specified network interface. The actual number of IP addresses added may be less than the total addresses if the IP address list is not large enough (which is set in the user configurable *NX_DHCP_IP_ADDRESS_MAX_LIST_SIZE* parameter).

Input Parameters

dhcp_ptr	Pointer to DHCP Server control block.
iface_index	Index corresponding to network interface
start_ip_address	First available IP address
end_ip_address	Last of the available IP address
addresses_added	Number of IP addresses added to list

Return Values

NX_SUCCESS	(0x00)	Successful DHCP Server create.
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_DHCP_BAD_INTERFACE_INDEX	(0xA1)	Index does not match addresses
NX_DHCP_INVALID_IP_ADDRESS_LIST	(0x99)	Illogical start/end addresses

Allowed From

Application

Example

```
/* Create a pool of available IP addresses to assign.  */
status =  nx_dhcp_create_server_ip_list (&dhcp_server, iface_index,
            START_IP_ADDRESS_LIST,  END_IP_ADDRESS_LIST, &addresses_added);

/* If status is NX_SUCCESS a IP address list was successfully created.
   addresses_added indicates how many IP addresses were actually added to the
   list. */
```

nx_dhcp_clear_client_record

Remove Client record from Server database

Prototype

```
UINT nx_dhcp_clear_client_record (NX_DHCP_SERVER *dhcp_ptr,
                                  NX_DHCP_CLIENT *dhcp_client_ptr);
```

Description

This service clears the Client record from the Server database.

Input Parameters

dhcp_ptr	Pointer to DHCP Server control block.
dhcp_client_ptr	Pointer to DHCP Client to remove

Return Values

NX_SUCCESS	(0x00)	Successful DHCP Server create.
NX_PTR_ERROR	(0x16)	Invalid pointer input.

Allowed From

Application

Example

```
/* Remove Client record from the server database.  */
status =  nx_dhcp_clear_client_record (&dhcp_server, &dhcp_client_ptr);
/* If status is NX_SUCCESS the specified Client was removed from the database. */
```

nx_dhcp_set_interface_network_parameters

Set network parameters for DHCP options

Prototype

```
UINT nx_dhcp_set_interface_network_parameters(NX_DHCP_SERVER *dhcp_ptr,
                        UINT iface_index, ULONG subnet_mask,
                        ULONG default_gateway_address,
                        ULONG dns_server_address);
```

Description

This service sets default values for network critical parameters for the specified interface. The DHCP server will include these options in its OFFER and ACK replies to the DHCP Client. If the host set interface parameters on which a DHCP server is running, the parameters will defaulted as follows: the router set to the primary interface gateway for the DHCP server itself, the DNS server address to the DHCP server itself, and the subnet mask to the same as the DHCP server interface is configured with.

Input Parameters

dhcp_ptr	Pointer to DHCP Server control block.
iface_index	Index corresponding to network interface
subnet_mask	Subnet mask for Client network
default_gateway_address	Client's router IP address
dns_server_address	DNS server for Client's network

Return Values

NX_SUCCESS	(0x00)	Successful DHCP Server create.
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_DHCP_BAD_INTERFACE_INDEX	(0xA1)	Index does not match addresses
NX_DHCP_INVALID_NETWORK_PARAMETERS	(0xA3)	Invalid network parameters

Allowed From

Application

Example

```
/* Set network parameters for a specific interface. */
status = nx_dhcp_set_interface_network_parameters(&dhcp_server, iface_index,
                NX_DHCP_SUBNET_MASK, NX_DHCP_DEFAULT_GATEWAY,
                NX_DHCP_DNS_SERVER);

/* If status is NX_SUCCESS network parameters were successfully set. */
```

nx_dhcp_server_delete

Delete a DHCP Server instance

Prototype

```
UINT nx_dhcp_server_delete(NX_DHCP_SERVER *dhcp_ptr);
```

Description

This service deletes a previously created DHCP Server instance.

Input Parameters

dhcp_ptr	Pointer to a DHCP Server instance.

Return Values

NX_SUCCESS	(0x00)	Successful DHCP Server delete.
NX_PTR_ERROR	(0x16)	Invalid pointer input.

Allowed From

Threads

Example

```
/* Delete a DHCP Server instance.  */
status =  nx_dhcp_server_delete(&dhcp_server);

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

nx_dhcp_server_start

Start DHCP Server processing

Prototype

```
UINT nx_dhcp_server_start(NX_DHCP_SERVER *dhcp_ptr);
```

Description

This service starts DHCP Server processing, which includes creating a server UDP socket, binding the DHCP port and waiting to receive Client DHCP requests.

Input Parameters

dhcp_ptr	Pointer to previously created DHCP instance.

Return Values

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

Allowed From

Threads

Example

```
/* Start the DHCP Server processing for this IP instance.  */
status =  nx_dhcp_server_start(&dhcp_server);

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

See Also

nx_dhcp_create, nx_dhcp_delete, nx_dhcp_release, nx_dhcp_state_change_notify, nx_dhcp_stop, nx_dhcp_user_option_retrieve, nx_dhcp_user_option_convert

nx_dhcp_server_stop

Stops DHCP Server processing

Prototype

```
UINT nx_dhcp_server_stop(NX_DHCP_SERVER *dhcp_ptr);
```

Description

This service stops DHCP Server processing, which includes of receiving DHCP Client requests.

Input Parameters

dhcp_ptr	Pointer to DHCP Server instance.

Return Values

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

Allowed From

Threads

Example

```
/* Stop the DHCP Server processing for this IP instance.  */
status =  nx_dhcp_server_stop(&dhcp_server);

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