Dynamic Host Configuration Protocol over IPv6
(DHCPv6 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 5.1

Contents

Chapter 1
Introduction to the NetX Duo DHCPv6Server

In IPv6 networks, DHCPv6is required for Clients to obtain IPv6 addresses. It does not replace DHCP which is limited to IPv4in that it does not offer IPv4 addresses. DHCPv6 has similar features to DHCP as well as many enhancements. AClient who does not or cannot use IPv6 stateless address auto-configuration can use DHCPv6 to be assigned a unique global IPv6 address from a DHCPv6 Server.

NetX Duo was developed by Expresslogic to support IPv6 network based applications and network protocols such as DHCPv6. This document will explain in detail how the NetX Duo DHCPv6 Server assigns IPv6 addressestoDHCPv6 Clients.

DHCPv6 Communication

DHCPv6 Message structure

Message content is basically a message header followed by one or more (usually more) option blocks. Below is the basic structure where each block represents one byte:

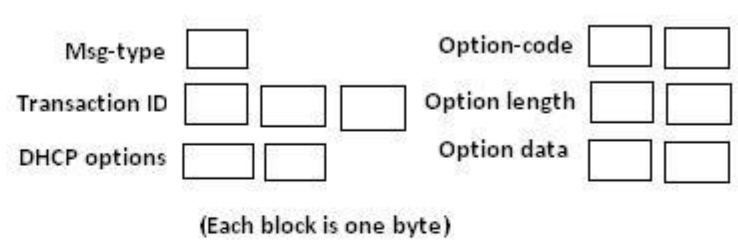

Figure 1. DHCPv6 message and option block structure

The 1-byte Msg-Type field indicates the type of DHCPv6 message. The 3-byte Transaction-ID field is set by the Client. It can by any sequence of characters but must be unique for each Client message to the Server (conserved across duplicate messages sent by the Client). The Server uses that Transaction-ID for each response to the Client to enable the Client to match up Server messages in the event of packets that are delayed or dropped on the network. Following the Transaction-ID field, are one or more DHCPv6 options used to indicate what the Client is requesting.

The DHCPv6 option structure is composed of an option code, an option length field, which does not include the length or code fields, and finally the option data itself which is one or more 2 byte option code fields for the data the Client is requesting.

Some option blocks have nested options. For example, an *Identity Association for Non Temporary Address (IANA)* option contains one or more *Identity Association (IA)* options to request IPv6 addresses. The *IANA* option returned in the Server Reply message contains the same *IA* option(s) with the IPv6 address and lease times granted by the Server, as well as a *Status* option indicating if there is an error with the Client address request.

A list of all option blocks and their description is provided in **Appendix A**.

DHCPv6 Message Types

Although DHCPv6 greatly enhances the functionality of DHCP, it uses the same number of messages as DHCP and supports the same vendor options as DHCP. The list of DHCPv6 messages are as follows:

SOLICT	(1)	(sent by Client)
ADVERTISE	(2)	(sent by Server)
REQUEST	(3)	(sent by Client)
REPLY	(7)	(sent by Server)
CONFIRM	(4)	(sent by Client)
RENEW	(5)	(sent by Client)
REBIND	(6)	(sent by Client)
RELEASE	(8)	(sent by Client)
DECLINE	(9)	(sent by Client)
INFORM_REQUEST	(11)	(sent by Client)
RECONFIGURE*	(10)	(sent by Server)

*RECONFIGURE is not supported by the NetX Duo DHCPv6 Server.

The basic DHCPv6 request sequence, with the equivalent DHCPv4 message type in parenthesis, is as follows:

Client ***Solicit*** (*Discovery*) → Server **Advertisement** (*Offer*) → Client **Request** (*Request*) → Server **Reply** (*DHCPAck*)

Client **Renew**(same) → Server **Reply** (*DHCPAck*)

DHCPv6 Message Validation

Transaction ID: The Client must generate a transaction ID for each message it sends to the Server. The DHCPv6 Server will reject any message from the Client not matching this transaction ID. The Server in turn must use the same transaction ID in its responses back to the Client.

DHCPv6 unique Identifiers (DUIDs)

All Server messages must also include a DHCPv6 unique Identifier (DUID) in each message or the DHCPv6 Client should not accept the message. A Link Layer (LL) DUID is a control block containing client MAC address, hardware type, and DUID type. A Link Layer Time (LLT) DUID additionally contains a time field which decreases the chances the DUID will not be unique on the host network. For that reason RFC 3315 recommends LLT DUIDs over LL DUIDs. If the host application does not create its own unique time value, NetX Duo DHCPv6 will provide a default one. The third type of DUID is the Enterprise (Vendor assigned) DUID which contains a registered Enterprise ID (as in registered with IANA) and private data that is variable in type and length e.g. based on memory size, operating system type of other hardware configuration. See the list of Configuration Options elsewhere in this document for setting up the Server vendor assigned and private ID values.

The Client must also include its DUID in its messages to the Server except for INFORM_REQUEST, or the Server will reject them.

DHCPv6 Client Server Sessions

DHCPv6 Clients and Servers exchange messages over UDP. The Client uses port 546 to send and receive DHCPv6 messages and the Server uses port 547. The Client initially uses its link-local address for transmitting and receiving DHCPv6 messages. It sends all messages to DHCPv6 servers using a reserved, link-scoped multicast address known as the *All_DHCP_Relay_Agents_and_Servers* multicast address (*FF02::01:02)***.**

ForIPv6 address assignment requests, the DHCPv6 Server listens for *Solicit* messages sent to the *All_DHCP_Relay_Agents_and_Servers* address. In the *Solicit* request, the Clientcan request the assignment of specific IPv6 address or let the Server choose one. It can also request other network configuration information from the Server.

If the DHCPv6Server extracts a valid *Solicit* message and can assign an IPv6 address to the Client, it responds with an *Advertise* message containing the IPv6 address it will grant to the Client, the IPv6 address lease time and any additional information requested by the Client. If the Client accepts the Server offer it responds with a *Request* message letting the Server know it will accept the IPv6 address. The Server confirms the Client is bound to the IPv6 address with a *Reply* message.

If the Client DHCPv6 message is invalid, the Server will discard the message silently. If the Server is unable to grant the request it will send a *Reply* message with an indication of the problem in the status field of the IP address IANA option. If duplicate Client requests are received the Server resends its previous DHCPv6 response, assuming the Client simply did not receive the packet.

It is up to the Client to verify that its assigned IPv6 address from the Server is not assigned to another host on the system by using various IPv6 protocols such as Duplicate Address Detection. If the address is not unique, the Client will send the Server a *Decline* message. The Server updates its IP lease table with this information, recording that the address is already assigned. Meanwhile the Client must restart the DHCPv6 request process with another *Solicit* message.

In addition to an IPv6 address, a Client will likely also need to know the DNS server and possibly other network information such as the network domain name. DHCPv6 provides the means to request this information using either the use of Option Requests in the *Solicit* and *Request* messages, or separately in *Information Request* messages. DHCPv6 options are explained later in this chapter.

IPv6 Lease Duration

When the Server grants an IPv6 address to a Client, it also assigns the lease duration (lifetime)in the IANA option for when it recommends the Client to start renewing (T1) or rebinding (T2) its IPv6 address using *Renew* and *Rebind* messages. The difference between the two is the Client directs the *Renew* message to the Server by including the Server DUID in the *Renew* request. However, it does not specify any server, and hence does not include a Server DUID, in the *Rebind* message to the *All_DHCP_Relay_Agents_and_Servers* address. The IA option which contains the actual IPv6 address the Server grants the Client also contains the preferred and valid lifetimes when the leased IPv6 address becomes deprecated or obsolete (invalid), respectively.

The NetX Duo DHCPv6 Server maintains a session timeout for each Client to track the time between Client messages. This is necessary in the event of a Client host losing connectivity or the network doing down. When the session timeout expires, it is assumed the Client is either no longer interested or able to make DHCPv6 requests of the Server. The Server deletes the Client record and returns any tentatively assigned IPv6 address back to the available pool. The session timeout wait is a user configurable option.

If the Client wishes to release its IPv6 address, or discovers that the IPv6 address assigned to it by the DHCPv6 Server is already in use, it send a *Release* or *Decline* message respectively. In the case of a *Release* message, the Server returns that IPv6 address status back to the available pool. In the case of the *Decline* message, it updates its IP lease table to indicate this IPv6 address is not available (owned by another entity elsewhere on the network).

IPv6 Lease and Client Record Data

When the DHCPv6 Server starts accepting Client requests it maintains a list of active Clients who are requesting or have been assigned IPv6 addresses. The Server checks for IP lease expiration by means of a lease timer that periodically updates the Client lease duration. When the duration exceeds the valid lifetime, the Server clears the Client record and returns its IPv6 address back to the available pool. It is up to the Client to start the renewal/rebinding process before this happens!

The NetX Duo DHCPv6 Server client record table contains information to identify Clients, and 'state' information for validating DHCPv6 Client requests and assigning or re-assigning IPv6 addresses. Such information includes:

- The Client DHCPv6 Unique Identifier (DUID) which uniquely defines each Client host on a network. The Client must always use this same DUID for all its DHCPv6 messages.

- The Client Identity Association for Non Temporary Addresses (IANA) and Identity Association IPv6 address (IA) cumulatively which define the Client IPv6 address assignment parameters.

- Client option requests (DNS server, domain name, etc).

- The Client IPv6 source address (if set) and destination address (if not multicast) of its most recent DHCPv6 request.

- The Client's most recent message type and DHCPv6 'state'.

NetX Duo DHCPv6 Server Requirements and Constraints

The NetX Duo DHCPv6 Server API requires ThreadX 5.1 or later, and NetX Duo 5.5 or later.

Requirements

IP Thread Task Setup

The NetX Duo DHCPv6 Server requires a creation of an IP instance for sending and receiving messages to DHCPv6 on its network link. This is done using the *nx_ip_create* service. The NetX Duo DHCPv6 Server itself must be created. This is done using the *nx_dhcpv6_server_create* service.

DHCPv6 utilizes NetX Duo, ICMPv6 and UDP. Therefore IPv6 must first be enabled prior to using DHCPv6 Server by calling the following NetX Duo services:

- *nx_udp_enable*
- *nxd_ipv6_enable*
- *nxd_icmp_enable*

Further, before the DHCPv6 Server can be started, it has a number of set up tasks to perform:

- Create and validate its link local and IPv6 global addresses. Address validation is performed automatically by NetX Duo using Duplicate Address Detection if it is enabled. See the *NetX Duo User Manual* for details on link local and global IP address validation.

- Set the network interface index for its DHCPv6 interface.

- Create an IP address range for assignable IPv6 addresses. Or, if data exists from a previous Server DHCPv6 session, IPv6 lease table and client records from that session must be uploaded from non volatile memory to the DHCPv6 Server. The small example system elsewhere in this document will demonstrate the DHCPv6 Server services for accomplishing this requirement.

- Set the Server DUID. If the Server has created its DUID in a previous session it must use the same data to create the same DUID for messages to its Clients. The small example system elsewhere in this document will demonstrate how this requirement is accomplished.

At this point the DHCPv6 Server is ready to run. Internally the NetX Duo DHCPv6 Server will create a UDP socket bound to port 547, and starts listening for Client requests.

Packet Pool Requirements

NetX Duo DHCPv6 Server requires a packet pool for sending DHCPv6 messages. The size of the packet pool in terms of packet payload and number of packets available is user configurable, and depends on the anticipated volume of DHCPv6 messages and other transmissions the host application will be sending.

A typical DHCPv6 message is about 200-300 bytes depending on the number of additional options requested by the Client, and information available from the Server.

Setting the DHCPv6 Server interface

The DHCPv6 Server defaults to the primary network interface as the interface it will accept Client requests on. However, the host application must still set the global address index which it used to create the Server global address. The DHCPv6 interface index and global address index are set using the *_nx_dhcpv6_server_interface_set*service. This is also demonstrated in the “small example” in this document.

Saving DHCPv6 DUID across Server Reboots

The DHCPv6 protocol requires the Server to use the same DUID across multiple reboots. Any data used to create the DUID must therefore be stored and retrieved from nonvolatile memory for this requirement. For hosts that use the Link Layer Plus Time DUID which requires access to a real time clock. The NetX Duo DHCPv6 host application should include real time data access for generating a time value for the initial Server DUID creation, and store that data for reuse on subsequent Server sessions. The *nx_dhcpv6_set_server_duid* then takes DUID data as its arguments, as well as configuration options depending on DUID type, to produce (or reproduce) its own DUID.

Assignable IPv6 Address List Creation

After creation of the DHCPv6 Server, the Server host application must create a range of assignable IPv6 global addresses if there is no previously stored IP address list data. This is done using the *nx_dhcpv6_create_ip_address_range*service which takes as input a starting and ending IPv6 address.

Saving DHCPv6 Assignable Address and Client data

The DHCPv6 protocol requires that the DHCPv6 Server save its Client and IPv6 address data in nonvolatile storage in the event of rebooting the server. The NetX Duo DHCPv6 Server has several API for uploading and downloading Client and IPv6 address data to and from the DHCPv6 Server, respectively:

_nx_dhcpv6_add_client_record
_nx_dhcpv6_add_ip_address_lease
_nx_dhcpv6_retrieve_client_record
_nx_dhcpv6_ip_address_lease

Uploading data to the Server must be done before restarting the Server. Downloading the data should be done only after the DHCPv6 Server is stopped (or suspended). The services for doing so are described in detail later in this document. However, the NetX Duo DHCPv6 provides does not define an access to nonvolatile storage. This must be handled by the host application. The small example demonstrates how the host application does this.

Server DHCP Unique Identifier (DUID)

The Server DUID uniquely defines the DHCPv6 Server host on the network. If a Server has not previously created its DUID, it can use the *nx_dhcpv6_server_set_duid* to create one. As per RFC 3315, the DHCPv6 Server must save this DUID to nonvolatile memory to be able to retrieve it after Server reboots. The DHCPv6 Server supports the Link Layer, Link Layer Time and Enterprise (Vendor assigned) DUID types. Note that the Client must send in the Vendor type DUID directly. The option for Vendor type DUIDs (17) is not directly supported by the NetX Duo DHPv6 Server.

The DHCPv6 Server host application has default values for IPv6 address assignment including lease timeout. See Configurable Options later in this document for how to set these options. :

The *IANA* control block contains the T1 and T2 fields. The *IA* block in the *IANA* control block contains the preferred and valid lifetime fields. The host application has configurable options defined elsewhere in this document for setting these options. They are assigned to all Client IPv6 address requests.

These DHCPv6 IP lease parameters are defined below.

T1 – time in seconds when the Client must start renewing its IPv6 address from the Server that assigned it.

T2 – time in seconds when the Client must start rebinding the IPv6 address, if renewal failed, with any Server on its link.

Preferred lifetime – time in seconds when the Client address becomes deprecated if the Client has not renewed or rebound it. The Client can still use this address.

Valid lifetime – time in seconds when the Client IP address is expired and MUST not use this address in its network transmissions..

The RFC recommends T1 and T2 times that are 0.5 and 0.8, respectively, of the preferred lifetime in the Client *IANA* option. If the Server has no preference, it should set these times to zero. If a Server reply contains T1 and T2 times set to zero, it is letting the Client set its own T1 and T2 times.

Constraints

NetX Duo DHCPv6 Server does not support the following DHCPv6 options:

- Rapid Commit option which optimizes the DHCPv6 address request process to just the Solicit and Reply message exchange
- Reconfigure option which allows the Server can initiate changes to the Client's IP address status.
- Unicast option; all Client messages must be sent to the *All_DHCP_Relay_Agents_and_Servers* multicast address rather than to the DHCPv6 Server directly.
- Identity Association for the Temporary Addresses(IA_TA)option which is a temporary IP address granted to a Client.
- Multiple IA (IPv6 addresses) options per Client Request
- Relay host between DHCPv6 Client and Server e.g. Client and Server must be on the same network.
- IPSec and Authentication are not supported in DHCPv6 messaging. However, the IP instance may be IPSec enabled depending on the version of NetX Duo in use.
- The NetX Duo DHCPv6 Server directly supports only the DNS server option request. This may change in future releases.
- The Prefix Delegation option is not supported.
- Authentication of DHCPv6 messages although IPSec can be enabled in the underlying NetX Duo environment. Neither does the NetX Duo DHCPv6 Server support relay connections to the Clients. It is assumed all Client requests originate from hosts on the Server network.

NetX Duo DHCPv6 Server Callback Functions

nx_dhcpv6_address_declined_handler

When the DHCPv6 Client sends a Decline message, the NetX Duo DHCPv6 Server marks the address as not available in its IPv6 address tables. To customize the Server handling of this message, the *nx_dhcpv6_address_declined_handler*is provided.

nx_dhcpv6_option_request_handler

When the DHCPv6 Client message contains option request data, the NetX Duo DHCPv6 Server forwards each option request option code to this user callback, if defined, to fill in the data. The Server will automatically fill in the DNS Server callback if requested.

.

Supported DHCPv6 RFCs

NetX Duo DHCPv6 is compliant with RFC3315, RFC3646, and related RFCs.

Chapter 2

Installation and Use of the NetX Duo DHCPv6 Server

This chapter contains a description of various issues related to installation, setup, and usage of the NetX Duo DHCPv6Server.

Product Distribution

The NetX Duo DHCPv6 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:

nxd_dhcpv6_server.h	NetX DuoDHCPv6Server header file
nxd_dhcpv6_server.c	NetX DuoDHCPv6Server source file
demo_netxduo_dhcpv6.c	NetX Duo DHCPv6 Server demo file
nxd_dhcpv6_server.pdf	NetX Duo DHCPv6Server User Guide

NetX Duo DHCPv6ServerInstallation

In order to use the NetX Duo DHCPv6Server API, the entire distribution mentioned previously should be copied to the same directory where NetX Duo is installed. For example, if NetX Duo is installed in the directory "*\threadx\arm7\green*" then the *nxd_dhcpv6_server.h* and *nx_dhpcv6_server.c* files should be copied into this directory.

Using NetX Duo DHCPv6 Server

Using the NetX Duo DHCPv6Server API is easy. Basically, the application code must include *nx_dhcpv6-server.h* after it includes *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX Duo, respectively. The application must also include *nxd_dhcpv6_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. This is all that is required to use NetX Duo DHCPv6 Server.

Note that since DHCPv6is based on the IPv6 protocol, IPv6 must be enabled on the IP instance using *nxd_ipv6_enable*. NetX Duo UDP and ICMPv6 services are also utilized. UDP is enabled by calling *nx_udp_enable* and ICMPv6is enabled by calling *nxd_icmp_enable* prior to starting the NetX Duo DHCPv6 Server thread task.

Small Example System

An example of how easy it is to use the NetX Duo DHCPv6 Server is described in the small example below using a DHCPv6 Client and Server running over a virtual “RAM” driver. This demo assumes a single homed host using the NetX Duo environment.

tx_application_define creates packet pool for sending DHCPv6 message, a thread and an IP instance for both the Client and Server, and enables UDP (DHCP runs over UDP) and ICMP for both Client and Server IP tasks in lines 89-166.

The DHCPv6 Server is created in line 192. It does not define the optional address decline or option request handlers. In the Server thread entry function, the Server IP is set up with a link local address and enabled for IPv6 and ICMPv6 services in lines 537-580.

Before starting the DHCPv6 Server, the host application creates a Server DUID in line 604 and sets the local network DNS server on line 589. It then creates a table of assignable IP addresses in lines 615 - 633. See the **Advanced Example System** in Appendix D for how to store and retrieve Server tables from memory.

Then the DHCPv6 Server is ready to start on line 636.

For details on creating and running the NetX Duo DHCPv6 Client see the *nxd_dhcpv6_client.pdf* file distributed on with the DHCPv6 Server.

```
1    /* This is a small demo of the NetX Duo DHCPv6 Client and Server for the high-performance
2       NetX Duo stack.  */
3
4    #include    <stdio.h>
5    #include   "tx_api.h"
6    #include   "nx_api.h"
7    #include   "nxd_dhcpv6_client.h"
8    #include   "nxd_dhcpv6_server.h"
9
10   /* Verify NetX Duo version.  */
11   #if (((__NETXDUO_MAJOR_VERSION__  >= 5) && (__NETXDUO_MINOR_VERSION__  >= 6)))
12   #define  MULTIHOME_NETXDUO
13   #endif  /* NETXDUO VERSION check */
14
15   #define     DEMO_STACK_SIZE         2048
16
17
18   /* Define the ThreadX and NetX object control blocks...  */
19
20   NX_PACKET_POOL          pool_0;
21   TX_THREAD               thread_client;
22   NX_IP                   client_ip;
23   TX_THREAD               thread_server;
24   NX_IP                   server_ip;
25
26   /* Define the Client and Server instances. */
27
28   NX_DHCPV6               dhcp_client;
29   NX_DHCPV6_SERVER        dhcp_server;
```

```


/* Define some global flags. */

UINT g_declined = NX_FALSE;
UINT g_released = NX_FALSE;
UINT g_dhcp_failed = NX_FALSE;
UINT g_client_bound = NX_FALSE;

/* Define a counter for DHCP state changes. */
UINT   state_changes;
/* Define the error counter used in the demo application...  */
ULONG  error_counter;

/* Define thread prototypes.  */

void    thread_client_entry(ULONG thread_input);
void    thread_server_entry(ULONG thread_input);
void    dhcpv6_state_change_notify(NX_DHCPV6 *dhcp_ptr, UINT old_state, UINT new_state);

/***** Substitute your ethernet driver entry function here *********/
VOID    _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);


/* Define some DHCPv6 parameters. */

#define DHCPV6_IANA_ID  0xC0DEDBAD
#define DHCPV6_T1       NX_DHCPV6_INFINITE_LEASE
#define DHCPV6_T2       NX_DHCPV6_INFINITE_LEASE


/* Declare NetX DHCPv6 Client callbacks. */

VOID dhcpv6_deprecated_IP_address_handler(NX_DHCPV6 *dhcpv6_ptr);
VOID dhcpv6_expired_IP_address_handler(NX_DHCPV6 *dhcpv6_ptr);


/* Define main entry point.  */

int main()
{

    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();
}


/* Define what the initial system looks like.  */

void    tx_application_define(void *first_unused_memory)
{

CHAR    *pointer;
UINT    status;

    /* Setup the working pointer.  */
    pointer =  (CHAR *) first_unused_memory;

    /* Create the Client thread.  */
    status = tx_thread_create(&thread_client, "Client thread", thread_client_entry, 0,
            pointer, DEMO_STACK_SIZE,
            8, 8, TX_NO_TIME_SLICE, TX_AUTO_START);
    /* Check for IP create errors.  */
    if (status)
    {
        error_counter++;
        return;
    }

    pointer =  pointer + DEMO_STACK_SIZE;

    /* Create the Server thread.  */
```

```
    status = tx_thread_create(&thread_server, "Server thread", thread_server_entry, 0,
            pointer, DEMO_STACK_SIZE,
            4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
    /* Check for IP create errors.  */
    if (status)
    {
        error_counter++;
        return;
    }

    pointer =  pointer + DEMO_STACK_SIZE;

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

    /* Create a packet pool.  */
    status =  nx_packet_pool_create(&pool_0, "NetX Main Packet Pool", 1024, pointer,
                                        NX_DHCPV6_PACKET_POOL_SIZE);
    pointer = pointer + NX_DHCPV6_PACKET_POOL_SIZE;

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

    /* Create a Client IP instance.  */
    status = nx_ip_create(&client_ip, "Client IP", IP_ADDRESS(0, 0, 0, 0),
                          0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
                          pointer, 2048, 1);

    pointer =  pointer + 2048;

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

    /* Create a Server IP instance.  */
    status = nx_ip_create(&server_ip, "Server IP", IP_ADDRESS(1, 2, 3, 4),
                          0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
                          pointer, 2048, 1);

    pointer =  pointer + 2048;

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

    /* Enable UDP traffic for sending DHCPv6 messages.  */
    status =  nx_udp_enable(&client_ip);
    status +=  nx_udp_enable(&server_ip);

    /* Check for UDP enable errors.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Enable ICMP.  */
    status =  nx_icmp_enable(&client_ip);
    status +=  nx_icmp_enable(&server_ip);

    /* Check for ICMP enable errors.  */
    if (status)
    {
        error_counter++;
        return;
```

```
    }

    /* Create the DHCPv6 Client. */
    status =  nx_dhcpv6_client_create(&dhcp_client, &client_ip, "DHCPv6 Client", &pool_0, pointer,
                                        NX_DHCPV6_THREAD_STACK_SIZE,
                                        dhcpv6_state_change_notify, NX_NULL,
                                        dhcpv6_deprecated_IP_address_handler,
                                        dhcpv6_expired_IP_address_handler);

    /* Check for errors.  */
    if (status)
    {
        error_counter++;
        return;
    }

    /* Update the stack pointer because we need it again. */
    pointer = pointer + NX_DHCPV6_THREAD_STACK_SIZE;

    /* Create the DHCPv6 Server. */
    status =  nx_dhcpv6_server_create(&dhcp_server, &server_ip, "DHCPv6 Server", &pool_0, pointer,
                                      NX_DHCPV6_SERVER_THREAD_STACK_SIZE, NX_NULL, NX_NULL);

    /* Check for errors.  */
    if (status != NX_SUCCESS)
    {
        error_counter++;
    }

    /* Update the stack pointer in case we need it again. */
    pointer = pointer + NX_DHCPV6_SERVER_THREAD_STACK_SIZE;

    /* Enable the Server IP for IPv6 and ICMPv6 services. */
    nxd_ipv6_enable(&server_ip);
    nxd_icmp_enable(&server_ip);

    /* Yield control to DHCPv6 threads and ThreadX. */
    return;
}


/* Define the Client host application thread. */

void    thread_client_entry(ULONG thread_input)
{

UINT        status;
NXD_ADDRESS ipv6_address;
ULONG       T1, T2, preferred_lifetime, valid_lifetime;
UCHAR       buffer[200];
USHORT      option_code;


    state_changes = 0;

    /* Establish the link local address for the host. The RAM driver creates
       a virtual MAC address of 0x1122334456. */
#ifdef MULTIHOME_NETXDUO
    status = nxd_ipv6_address_set(&client_ip, 0, NX_NULL, 10, NULL);
#else
    status = nxd_ipv6_linklocal_address_set(&client_ip, NULL);
#endif

    /* Let NetX Duo and the network driver get initialized. Also give the server time to get set up.
*/
    tx_thread_sleep(300);

    /* Enable the Client IP for IPv6 and ICMPv6 services. */
    nxd_ipv6_enable(&client_ip);
    nxd_icmp_enable(&client_ip);

    /* Create a Link Layer Plus Time DUID for the DHCPv6 Client. Set time ID field
```

```
       to NULL; the DHCPv6 Client API will supply one. */
    status = nx_dhcpv6_create_client_duid(&dhcp_client, NX_DHCPV6_DUID_TYPE_LINK_TIME,
                                          NX_DHCPV6_HW_TYPE_IEEE_802, 0);

    if (status != NX_SUCCESS)
    {
        error_counter++;
        return;
    }


    /* Create the DHCPv6 client's Identity Association (IA-NA) now.

       Note that if this host had already been assigned in IPv6 lease, it
       would have to use the assigned T1 and T2 values in loading the DHCPv6
       client with an IANA block.
    */
    status = nx_dhcpv6_create_client_iana(&dhcp_client, DHCPV6_IANA_ID, DHCPV6_T1,  DHCPV6_T2);

    if (status != NX_SUCCESS)
    {
        error_counter++;
        return;
    }

    memset(&ipv6_address,0x0, sizeof(NXD_ADDRESS));
    ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;

    /* Create an IA address option.

        Note that if this host had already been assigned in IPv6 lease, it
        would have to use the assigned IPv5 address, preferred and valid lifetime
        values in loading the DHCPv6 Client with an IA block.
    */
    status = nx_dhcpv6_create_client_ia(&dhcp_client, &ipv6_address, NX_DHCPV6_RENEW_TIME,
                                        NX_DHCPV6_REBIND_TIME);

    if (status != NX_SUCCESS)
    {
        error_counter++;
        return;
    }

    /* Starting up the NetX DHCPv6 Client. */

    /* If the host has no IPv6 address assigned, set the time expired to zero.

       If the host has been assigned an IPv6 lease, the host needs to supply time expired on the
       lease since stopping the Client when starting the Client back up. This may require access
       to a real time clock or other component. */

    /* Start the NetX DHCPv6 Client.  */
    status =  nx_dhcpv6_start(&dhcp_client, 0);

    /* Check for errors. */
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
    }

    /* Set the list of desired options to enabled. */
    nx_dhcpv6_request_option_timezone(&dhcp_client, NX_TRUE);
    nx_dhcpv6_request_option_DNS_server(&dhcp_client, NX_TRUE);
    nx_dhcpv6_request_option_time_server(&dhcp_client, NX_TRUE);
    nx_dhcpv6_request_option_domain_name(&dhcp_client, NX_TRUE);

    /* No, so the host should solicit a DHCPv6 server who will assign it one. */
    status = nx_dhcpv6_request_solicit(&dhcp_client);

    /* Check status.  */
```

```
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
    }

    /* Use this service to query the DHCPv6 Client for assigned address and network data.
       Alternatively the host application can wait for the g_client_bound signal to be set
       by the address change notify callback (see below). */
    do
    {
        /* Check if the DHCP Client is assigned an address yet. */
         status = nx_dhcpv6_get_IP_address(&dhcp_client, &ipv6_address);

         /* Check if we got a signal the DHCP client failed to get an IP address,
            e.g. retry count exceeded without a response. */
         if (g_dhcp_failed)
         {
             /* It did. We will need to restart the DHCP Client. */
             break;
         }

         tx_thread_sleep(100);

    } while (status != NX_SUCCESS);

    /* Did we get an assigned address? */
    if (status == NX_SUCCESS)
    {

        /* Yes; Register the assigned IP address with NetX Duo.  This assumes a 64 bit prefix. */
        status = nx_dhcpv6_register_IP_address(&dhcp_client, &ipv6_address, 64);

        /* Check status.  */
        if (status != NX_SUCCESS)
        {

            error_counter++;
            return;
        }

        /* If duplicate address detection is enabled, give it time to verify this
           address is unique on our network. */
        tx_thread_sleep(400);

        /* Do stuff with our assigned address. */
        tx_thread_sleep(100);

        /* Get IP address lease time. */
        status = nx_dhcpv6_get_lease_time_data(&dhcp_client, &T1, &T2, &preferred_lifetime,
                                                    &valid_lifetime);

        /* Check status.  */
        if (status != NX_SUCCESS)
        {
            error_counter++;
        }

        /* Get non standard option data. */
        memset(buffer, 0, 200);

        /* Set the information request option desired from Client record. */
        option_code = NX_DHCPV6_DOMAIN_NAME_OPTION;

        /* Get the information request data from the Client. The address_status field indicates
           if the DHCPv6 Client IP address was successfully registered with the DHCP server:
           1 = VALID; 2 = INVALID indicating the rest of the option data never came through. */
        status = nx_dhcpv6_get_other_option_data(&dhcp_client, option_code, buffer);

        if ((status != NX_SUCCESS) || (strlen((const char *)buffer)==0))
        {
```

```
            error_counter++;
        }

        /**** If we detect another host with the same address, we should decline the address. If we
           did not, use the address till we are done with it (leaving the network) and release it.
         *****/

#if 1 /* Releasing! */

        /* Release the address back e.g. leaving the network. Send a message to
           the server we are releasing the assigned address.  */
        status = nx_dhcpv6_request_release(&dhcp_client);

        /* Check status.  */
        if (status != NX_SUCCESS)
        {

            error_counter++;
            return;
        }

        /* Wait for the signal the assigned address is released. */
        while (!g_released)
        {
            tx_thread_sleep(100);
        }

        /* Reset the release flag. */
        g_released = NX_FALSE;

#else  /* Declining! */

        /* If we think the address is not unique, we must decline it. */
        status = nx_dhcpv6_request_decline(&dhcp_client);

        /* Check status.  */
        if (status != NX_SUCCESS)
        {

            error_counter++;
            return;
        }

        /* Wait for the signal the server knows the assigned address is declined. */
         while (!g_declined)
         {
             tx_thread_sleep(100);
         }

         /* Reset the decline flag. */
         g_declined = NX_FALSE;

#endif
    }

    /* Stopping the Client task. */
    status = nx_dhcpv6_stop(&dhcp_client);
    /* Check status.  */
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
    }

    /* Clearing out the old session. */
    status = nx_dhcpv6_reinitialize(&dhcp_client);
    /* Check status.  */
    if (status != NX_SUCCESS)
    {

        error_counter++;
```

```
        return;
    }

    /* Now request previously assigned address. This is a hint to the server. */
    status = nx_dhcpv6_create_client_ia(&dhcp_client, &ipv6_address, NX_DHCPV6_RENEW_TIME,
                                        NX_DHCPV6_REBIND_TIME);
    /* Check status.  */
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
    }

    /* Starting up the Client task. */
    status = nx_dhcpv6_start(&dhcp_client, 0);
    /* Check status.  */
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
    }

    /* Soliciting an IP address with a 'hint' of what address we'd like. */
    status = nx_dhcpv6_request_solicit(&dhcp_client);
    /* Check status.  */
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
    }

    do
    {

         status = nx_dhcpv6_get_IP_address(&dhcp_client, &ipv6_address);

         tx_thread_sleep(100);

    } while (status != NX_SUCCESS);


    /* Wait a bit before releasing the IP address and terminating the client. */
    tx_thread_sleep(100);

    /* Ok, lets stop the host application. In this case we DO NOT plan
       to keep the IPv6 address we were assigned to release it
       back to the DHCPv6 server. */
    status = nx_dhcpv6_request_release(&dhcp_client);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {
        error_counter++;
    }


    /* Now delete the DHCPv6 client and release ThreadX and NetX resources back to
       the system. */
    nx_dhcpv6_client_delete(&dhcp_client);

    return;

}

/* Define the test server thread. */
void    thread_server_entry(ULONG thread_input)
{

UINT        status;
```

```
NXD_ADDRESS ipv6_address_primary, dns_ipv6_address;
ULONG       duid_time;
UINT        addresses_added;
NXD_ADDRESS start_ipv6_address;
NXD_ADDRESS end_ipv6_address;


    /* Wait till the IP task thread has had a chance to set the device MAC address. */
    tx_thread_sleep(100);

    /* Make the Server IP IPv6 and ICMPv6 enabled. */
    nxd_ipv6_enable(&server_ip);
    nxd_icmp_enable(&server_ip);

    memset(&ipv6_address_primary,0x0, sizeof(NXD_ADDRESS));

    ipv6_address_primary.nxd_ip_version = NX_IP_VERSION_V6 ;
    ipv6_address_primary.nxd_ip_address.v6[0] = 0x20010db8;
    ipv6_address_primary.nxd_ip_address.v6[1] = 0xf101;
    ipv6_address_primary.nxd_ip_address.v6[2] = 0x00000000;
    ipv6_address_primary.nxd_ip_address.v6[3] = 0x00000101;

    /* Set the link local and global addresses. */

#ifndef NETXDUO_MULTIHOME_SUPPORT

    status = nxd_ipv6_linklocal_address_set(&server_ip, NULL);

    status += nxd_ipv6_global_address_set(&server_ip, &ipv6_address_primary, 64);


#else

    status = nxd_ipv6_address_set(&server_ip, 0, NX_NULL, 10, NULL);

    status += nxd_ipv6_address_set(&server_ip, 0, &ipv6_address_primary, 64, NULL);

#endif  /* NETXDUO_MULTIHOME_SUPPORT */

    /* Check for errors. */
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
    }

    /* Note this example assumes a single global IP address on the primary interface.  If otherwise
       the host should call the service to set the network interface and global IP address index.

        UINT _nx_dhcpv6_server_interface_set(NX_DHCPV6_SERVER *dhcpv6_server_ptr, UINT
                                             interface_index, UINT address_index)
    */

    /* Validate the link local and global addresses. */
    tx_thread_sleep(500);

    /* Set up the DNS IPv6 server address. */
    dns_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    dns_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
    dns_ipv6_address.nxd_ip_address.v6[1] = 0x0000f101;
    dns_ipv6_address.nxd_ip_address.v6[2] = 0x00000000;
    dns_ipv6_address.nxd_ip_address.v6[3] = 0x00000107;

    status = nx_dhcpv6_create_dns_address(&dhcp_server, &dns_ipv6_address);

    /* Check for errors. */
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
```

```
    }

     /* Note: For DUID types that do not require time, the 'duid_time' input can be left at zero.
        The DUID_TYPE and HW_TYPE are configurable options that are user defined in
        nx_dhcpv6_server.h.  */

    /* Set the DUID time as the start of the millenium. */
    duid_time = SECONDS_SINCE_JAN_1_2000_MOD_32;
    status = nx_dhcpv6_set_server_duid(&dhcp_server,
                                    NX_DHCPV6_SERVER_DUID_TYPE, NX_DHCPV6_SERVER_HW_TYPE,
                                    dhcp_server.nx_dhcpv6_ip_ptr -> nx_ip_arp_physical_address_msw,
                                    dhcp_server.nx_dhcpv6_ip_ptr -> nx_ip_arp_physical_address_lsw,
                                    duid_time);
    if (status != NX_SUCCESS)
    {
        error_counter++ ;
        return;
    }

    start_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    start_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
    start_ipv6_address.nxd_ip_address.v6[1] = 0x00000f101;
    start_ipv6_address.nxd_ip_address.v6[2] = 0x0;
    start_ipv6_address.nxd_ip_address.v6[3] = 0x00000110;

    end_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    end_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
    end_ipv6_address.nxd_ip_address.v6[1] = 0x0000f101;
    end_ipv6_address.nxd_ip_address.v6[2] = 0x00000000;
    end_ipv6_address.nxd_ip_address.v6[3] = 0x00000120;

    status = nx_dhcpv6_create_ip_address_range(&dhcp_server, &start_ipv6_address, &end_ipv6_address,
                                                    &addresses_added);

    if (status != NX_SUCCESS)
    {
        error_counter++ ;
        return;
    }

    /* Start the NetX DHCPv6 server!  */
    status =  nx_dhcpv6_server_start(&dhcp_server);

    /* Check for errors. */
    if (status != NX_SUCCESS)
    {
        error_counter++;
    }

    /* Server is running! */
    return;
}

/* Start of DHCPv6 Client callbacks. */

/*  This is an optional user defined callback the NetX DHCPv6 Client will call when it
    detects the Client task has changed state. */
void dhcpv6_state_change_notify(NX_DHCPV6 *dhcpv6_ptr, UINT old_state, UINT new_state)
{

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

    switch (old_state)
    {
        case NX_DHCPV6_STATE_SENDING_REQUEST:
        {
            /* Check if the request failed and client state set back to INIT. */
            if (new_state == NX_DHCPV6_STATE_INIT)
            {

                g_dhcp_failed = NX_TRUE;
```

```
            }

            return;

        }
        case NX_DHCPV6_STATE_SENDING_RENEWAL:
        {
            /* Check if the request failed and client state set back to INIT. */
            if (new_state == NX_DHCPV6_STATE_INIT)
            {
                /* Address still valid; use the assigned address until the valid timeout expires. */
            }

            return;
        }

        case NX_DHCPV6_STATE_SENDING_REBIND:
        {
            /* Check if the request failed and client state set back to INIT. */
            if (new_state == NX_DHCPV6_STATE_INIT)
            {
                /* Indicate that the rebind failed. */
                g_dhcp_failed = NX_TRUE;
            }

            return;

        }
        case NX_DHCPV6_STATE_SENDING_DECLINE:
        {

            /* Client address is declined!*/

            if (new_state == NX_DHCPV6_STATE_INIT)
            {
                g_declined = NX_TRUE;
            }

            break;
        }

        case NX_DHCPV6_STATE_SENDING_RELEASE:
        {

            /* Client address is released */

            if (new_state == NX_DHCPV6_STATE_INIT)
            {
                g_released = NX_TRUE;
            }

            break;
        }

        default:
           break;
    }

    if (new_state == NX_DHCPV6_STATE_BOUND_TO_ADDRESS)
    {
            /* Client is bound! Set the signal the Client has been
               assigned an IP address. Alternatively the host application
               can continue to poll the DHCPv6 client for a non zero IP address. */
            g_client_bound = NX_TRUE;
            break;
    }

    return;
}


/*  This is an optional user defined callback the NetX DHCPv6 Client will call when it
```

```
    detects the preferred lifetime of the Client IPv6 address has expired.  The Client
    IPv6 address is now deprecated.  */

VOID dhcpv6_deprecated_IP_address_handler(NX_DHCPV6 *dhcpv6_ptr)
{

    /* Call the renew request service. This just sets the state and returns.
       We will be notified if it succeeded by the state change handler. */
    nx_dhcpv6_request_renew(dhcpv6_ptr);

    return;
}

/*  This is an optional user defined callback the NetX DHCPv6 Client will call when it
    detects the valid lifetime of the Client IPv6 address has expired.  The Client
    IPv6 address is now expired.  */

VOID dhcpv6_expired_IP_address_handler(NX_DHCPV6 *dhcpv6_ptr)
{

    /* Call the rebind request service. This just sets the state and returns.
       We will be notified if it succeeded by the state change handler. */
    nx_dhcpv6_request_rebind(dhcpv6_ptr);

    return;
}
```

Figure 6. Example of the NetX Duo DHCPv6 Server

Chapter 3 NetX Duo DHCPv6 Server Configuration Options

There are several configuration options for building a NetX Duo DHCPv6 Server application. The following list describes each in detail:

Define	Meaning
NX_DISABLE_ERROR_CHECKING	This option removes DHCPv6 error checking. It it typically enabled when the application is debugged.
NX_DHCPV6_SERVER_THREAD_STACK_SIZE	This defines the size of the DHCPv6 thread stack. By default, the size is 4096 bytes which is more than enough for most NetX Duo applications.
NX_DHCPV6_SERVER_THREAD_PRIORITY	This defines the DHCPv6Server thread priority. This should be lower than the DHCPv6 Server's IP thread task priority. The default value is 2.
NX_DHCPV6_IP_LEASE_TIMER_INTERVAL	Timer interval in seconds when the lease timer entry function is called by the ThreadX scheduler. The entry function sets a flag for the DHCPv6 Server to increment all Clients' accrued time on their lease by the timer interval. By default, this value is 60.
NX_DHCPV6_SESSION_TIMER_INTERVAL	Timer interval insecondswhen the session timer entry function is called by the ThreadX scheduler. The entry function sets a flag for the DHCPv6 Server to incrementall active Client session time accrued by the timer interval. By default, this value is 3.

The following defines apply to the status option message type and the user configurable message. The status option indicates the outcome of a Client request:

NX_DHCPV6_STATUS_MESSAGE_SUCCESS
"IA OPTION GRANTED"

NX_DHCPV6_STATUS_MESSAGE_UNSPECIFIED
"IA OPTION NOT GRANTED-FAILURE UNSPECIFIED"

NX_DHCPV6_STATUS_NO_ADDRS_AVAILABLE
"IA OPTION NOT GRANTED-NO ADDRESSES AVAILABLE"

NX_DHCPV6_STATUS_MESSAGE_NO_BINDING
"IA OPTION NOT GRANTED-INVALID CLIENT REQUEST"

NX_DHCPV6_STATUS_MESSAGE_NOT_ON_LINK
"IA OPTION NOT GRANTED-CLIENT NOT ON LINK"

NX_DHCPV6_STATUS_MESSAGE_USE_MULTICAST
"IA OPTION NOT GRANTED-CLIENT MUST USE MULTICAST"

NX_DHCPV6_PACKET_WAIT_OPTION
This definesthe wait option for the Server *nx_udp_socket_receive* call. This is perfunctory since the socket has a receive notify callback from NetX Duo, so the packet is already enqueued when the DHCPv6 server calls the receive function. The default value is 100 ticks.

NX_DHCPV6_SERVER_DUID_TYPE This defines the Server DUID type which the Server includes in all messages to Clients. The default value is link layer plus time (NX_DHCPV6_SERVER_DUID_TYPE_LINK_TIME).

NX_DHCPV6_SERVER_HW_TYPE This defines the hardware type in the DUID link layer and link layer plus time options. The default value is NX_DHCPV6_HW_TYPE_IEEE_802.

NX_DHCPV6_PREFERENCE_VALUE This defines the preference option value between 0 and 255, where the higher the value the higher the preference, in the DHCPv6 option of the same name. This tells the Client what preference to place on this Server's offer where multiple DHCPv6 Servers are available to assign IP addresses. A value of 255 instructs the Client to choose this server. A value of zero indicates the Client is free to choose. The default value is zero.

NX_DHCPV6_MAX_OPTION_REQUEST_OPTIONS

This defines the maximum number of option requests in a Client request that can be saved to a Client record. The default value is 6.

NX_DHCPV6_DEFAULT_T1_TIME The time in seconds assigned by the Server on a Client address lease for when the Client should begin renewing its IP address. The default value is 2000 seconds.

NX_DHCPV6_DEFAULT_T1_TIME The time in seconds assigned by the Server on a Client address lease for when the Client should begin renewing its IP address. The default value is 2000 seconds.

NX_DHCPV6_DEFAULT_T2_TIME The time in seconds assigned by the Server on a Client address lease for when the Client should begin rebinding its IP address assuming its attempt to renew failed. The default value is 3000 seconds.

NX_DHCPV6_DEFAULT_PREFERRED_TIME

This defines the time in seconds assigned bythe Server for when an assigned Client IP address lease is deprecated. The default value is
2 *NX_DHCPV6_DEFAULT_T1_TIME.

NX_DHCPV6_DEFAULT_VALID_TIME

This defines the time expiration in seconds assigned by the Server on an assigned Client IP address lease. After this time expires, the Client IP address is invalid. The default value is 2 *NX_DHCPV6_DEFAULT_PREFERRED_TIME.

NX_DHCPV6_STATUS_MESSAGE_MAX

Defines the maximum size of the Server message in status option message field . The default value is 100 bytes.

NX_DHCPV6_MAX_LEASES

Defines the size of the Server's IP lease table (e.g. the max number of IPv6 address available to lease that can be stored). By default, this value is 100.

NX_DHCPV6_MAX_CLIENTS

Defines the size of the Server's Client record table (e.g. max number of Clients that can be stored).This value should be less than or equal to the value NX_DHCPV6_ MAX_LEASES. By default, this value is 120.

NX_DHCPV6_PACKET_TIME_OUT

Defines the wait option in timer ticks for the DHCPv6 Server wait on packet allocations. The default value is 3 * NX_DHCPV6_SERVER_TICKS_PER_SECOND.

NX_DHCPV6_PACKET_SIZE

Defines the size of the packet payload in the Server packet pool. The default value is 500 bytes.

NX_DHCPV6_PACKET_POOL_SIZE

Defines the size of the Server packet pool. The default value is 10* NX_DHCPV6_PACKET_SIZE (not quite 10 packets).

NX_DHCPV6_PACKET_RECEIVE_WAIT

Defines the wait option in packet allocate calls on the Server packet pool. The default value is (3 * NX_DHCPV6_SERVER_TICKS_PER_SECOND) , or 3 seconds.

NX_DHCPV6_PACKET_SIZE	This defines the packet payload of the Server packet pool packets. The default value is 500 bytes.
NX_DHCPV6_PACKET_POOL_SIZE	Defines the Server packet pool size for packets the Server will allocate to send DHCPv6 messages out. The default value is (10 * NX_DHCPV6_PACKET_SIZE).
NX_DHCPV6_TYPE_OF_SERVICE	This defines the type of service for UDP packet transmission. By default, this value is NX_IP_NORMAL.
NX_DHCPV6_FRAGMENT_OPTION	This defines the Server socket fragmentation option. The default value is NX_DON'T_FRAGMENT
NX_DHCPV6_TIME_TO_LIVE	Specifies the number of routers DHCPv6 packets from the Server may 'hop' pass before packets are discarded. The default value is set to 0x80.
NX_DHCPV6_QUEUE_DEPTH	Specifies the number of packets to keep in the Server UDP socket receive queue before NetX Duo discards packets.

Chapter 4 NetX Duo DHCPv6Server Services

This chapter contains a description of all NetX Duo DHCPv6Server services (listed below).

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_dhcpv6_server_create
Create a DHCPv6serverinstance
nx_dhcpv6_server_delete
Delete a DHCPv6serverinstance
nx_dhcpv6_server_start
Start the DHCPv6 server task
nx_dhcpv6_server_suspend
Suspend the DHCPv6 server task
nx_dhcpv6_server_resume
Resume DHCPv6 client processing
nx_dhcpv6_create_dns_address
Set the DNS server for option requests
nx_dhcpv6_create_ip_address_range
Create the range of IP addresses to lease
nx_dhcpv6_reserve_ip_address_range
Reserve range of IP addresses in server list
nx_dhcpv6_set_server_duid
Set the Server DUID for DHCPv6 packets
nx_dhcpv6_add_ip_address_lease
Add a lease record to the DHCPv6 server table
Nx_dhcpv6_retrieve_ip_address_lease
Retrieve an IP lease record from the Server table
nx_dhcpv6_add_client_record
Add a DHCPv6 Client record to the Server table
nx_dhcpv6_retrieve_client_record
Retrieve a client record from the Server table
Nx_dhcpv6_server_interface_set
Set the interface index for Server DHCPv6 services

nx_dhcpv6_create_dns_address

Set the network DNS server

Prototype

```
UINT  nx_dhcpv6_create_dns_address(
             NX_DHCPV6_SERVER *dhcpv6_server_ptr,
             NXD_ADDRESS *dns_ipv6_address);
```

Description

This service loads the DHCPv6 Server with the DNS server address for the Server DHCPv6 network interface.

Input Parameters

dhcpv6_server_ptr Pointer to DHCPv6 Server
dns_ipv6_address Pointer to the DNS server

Return Values

NX_SUCCESS	(0x00)	DNS Serversaved to DHCPv6 Server instance
NX_DHCPV6_INVALID_INTERFACE_IP_ADDRESS	(0xE95)	An invalid address is supplied
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Application Code

Example

```
    /* Set the network DNS server with the input address for the Server DHCPv6interface. */
status = nx_dhcpv6_create__dns_address(&dhcp_server_0, &dns_ipv6_address);
    /* If this service returns NX_SUCCESS the DNS server data was accepted. */
```

See Also

nx_dhcpv6_create_ip_address_range, nx_dhcpv6_server_create, nx_dhcpv6_reserve_ip_address_range

nx_dhcpv6_create_ip_address_range

Create the Server IP address list

Prototype

```
UINT _nx_dhcpv6_create_ip_address_range(
      NX_DHCPV6_SERVER *dhcpv6_server_ptr,
      NXD_ADDRESS *start_ipv6_address, NXD_ADDRESS *end_ipv6_address,
      UINT *addresses_added)
```

Description

This service creates the IP address list specified by the start and end addresses of the Server's assignable address range. The start and end addresses must match the Server interface address prefix (must be on the same link as the Server DHCPv6 interface). The number of addresses actually added is returned.

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server
start_ipv6_address	Start of addresses to add
end_ipv6_address	End of addresses to add
***addresses_added**	Output of addresses added

Return Values

NX_SUCCESS	(0x00)	IP address list successfully created
NX_DHCPV6_INVALID_INTERFACE_IP_ADDRESS	(0xE95)	An invalid address is supplied
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Application Code

Example

```
/* Create the Server IP address list for the server DHCPv6 interface. */

status = nx_dhcpv6_create_ip_address_range(&dhcp_server_0,
       &start_ipv6_address, &end_ipv6_address, &addresses_reserved);

/* If status is NX_SUCCESS one or more addresses were successfully added.  */
```

See Also

nx_dhcpv6_create_interface_dns_address, nx_dhcpv6_server_create, nx_dhcpv6_reserve_ip_address_range

nx_dhcpv6_reserve_ip_address_range

Reserve specified range of IP addresses

Prototype

```
UINT _nx_dhcpv6_reserve_ip_address_range(
      NX_DHCPV6_SERVER *dhcpv6_server_ptr,
      NXD_ADDRESS *start_ipv6_address, NXD_ADDRESS *end_ipv6_address,
      UINT *addresses_reserved)
```

Description

This service reserves the IP address range specified by the start and end addresses. These addresses must be within in the previously created server IP address range. These addresses will not be assigned to any Clients by the DHCPv6 Server. The start and end addresses must match the Server interface address prefix (must be on the same link as the Server DHCPv6 network interface). The number of addresses actually reserved is returned.

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server
start_ipv6_address	Start of addresses to reserve
end_ipv6_address	End of addresses to reserve
***addresses_reserved**	Number of addresses reserved

Return Values

NX_SUCCESS	(0x00)	RELEASE message successfully created and processed
NX_DHCPV6_INVALID_INTERFACE_IP_ADDRESS	(0xE95)	An invalid address is supplied
NX_DHCPV6_INVALID_IP_ADDRESS	(0xED1)	Starting address not found in Server address list.
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Application Code

Example

```
/* Reserve a range of ip addresses in the Server address table for the server DHCPv6
   network interface. */

status = nx_dhcpv6_reserve_ip_address_range(&dhcp_server_0,
        &start_ipv6_address, &end_ipv6_address, &addresses_reserved);

/* If status is NX_SUCCESS one or more addresses were successfully reserved.  */
```

See Also

nx_dhcpv6_create_interface_dns_address, nx_dhcpv6_server_create, nx_dhcpv6_create_ip_address_range

nx_dhcpv6_server_create

Create the DHCPv6 Server instance

Prototype

```
UINT   nx_dhcpv6_server_create(NX_DHCPV6_SERVER *dhcpv6_server_ptr,
                         NX_IP *ip_ptr, CHAR *name_ptr,
                         NX_PACKET_POOL *packet_pool_ptr,
                         VOID *stack_ptr,ULONG stack_size,
      VOID (*dhcpv6_address_declined_handler)(struct
                         NX_DHCPV6_SERVER_STRUCT *dhcpv6_server_ptr,
                         NX_DHCPV6_CLIENT *dhcpv6_client_ptr,
                         UINT message),
      VOID (*dhcpv6_option_request_handler)(
            struct NX_DHCPV6_SERVER_STRUCT *dhcpv6_server_ptr,
            UINT option_request, UCHAR *buffer_ptr, UINT *index));
```

Description

This service creates the DHCPv6 Server task with the specified input. The callback handlers are optional input. The stack pointer, IP instance and packet pool input are required. The IP instance and packet pool must already be created.

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server
ip_ptr	Pointer to the IP instance
name_str	Pointer to Server name
packet_pool_ptr	Pointer to Server packet pool
stack_ptr	Pointer to Server stack memory
stack_size	Size of Server stack memory
dhcpv6_address_declined_handler	Pointer to Client Decline or Release message handler
dhcpv6_option_request_handler	Pointer to options request option handler

Return Values

NX_SUCCESS	(0x00)	Server successfully resumed
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR		Invalid non pointer input

Allowed From

Application Code

Example

```
/* Create the DHCPv6 Server. */
status =  nx_dhcpv6_server_create(&dhcp_server_0, &ip_0, "DHCPv6 Server",
                                    &pool_0, stack_pointer,2048,
                                    dhcpv6_decline_handler,
                                     dhcpv6_get_time_handler);
```

```
/* If status is NX_SUCCESS the Server successfully created.  */
```

See Also
nx_dhcpv6_server_delete

nx_dhcpv6_server_delete

Delete the DHCPv6 Server

Prototype

```
UINT _nx_dhcpv6_server_delee(NX_DHCPV6_SERVER *dhcpv6_server_ptr)
```

Description

This service deletes the DHCPv6 Server task and any request that the Server was processing.

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server

Return Values

NX_SUCCESS	(0x00)	Server successfully deleted
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Delete the DHCPv6 Serve.  */
status = nx_dhcpv6_server_delete(&dhcp_server_0);

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

See Also

nx_dhcpv6_server_create

nx_dhcpv6_server_resume

Resume DHCPv6 Server task

Prototype

```
UINT _nx_dhcpv6_server_resume(NX_DHCPV6_SERVER *dhcpv6_server_ptr)
```

Description

This service resumes the DHCPv6 Server task and any request that the Server was processing.

Input Parameters

dhcpv6_server_ptr Pointer to DHCPv6 Server

Return Values

NX_SUCCESS (0x00) Server successfully resumed
NX_DHCPV6_ALREADY_STARTED
(0xE91) Server is running already
status(variable) ThreadX and NetX Duo error status
NX_PTR_ERROR (0x16) Invalid pointer input

Allowed From

Threads

Example

```
/* Resume the DHCPv6 Server task.  */
status = nx_dhcpv6_server_resume(&dhcp_server_0);

/* If status is NX_SUCCESS the Server successfully resumed.  */
```

See Also

nx_dhcpv6_server_start, nx_dhcpv6_server_suspend

nx_dhcpv6_server_start

Start the DHCPv6 Server task

Prototype

```
UINT _nx_dhcpv6_server_start(NX_DHCPV6_SERVER *dhcpv6_server_ptr)
```

Description

This service starts the DHCPv6 Server task and readies the Server to process application requests for receiving DHCPv6 Client messages. It verifies the Server instance has sufficient information (Server DUID), creates and binds the UDP socket for sending and receiving DHCPv6 messages, and activates timers for keeping track of session time and IP lease expiration.

Note: Before the DHCPv6 Server can run, the host application is responsible for creating the IP address range from which the Server can assign IP addresses. It is also responsible for setting the Server DUID and DHCPv6 interface (see *nx_dhcpv6_server_duid_set* and *nx_dhcpv6_server_interface_set* respectively.

Input Parameters

dhcpv6_server_ptr Pointer to DHCPv6 Server

Return Values

NX_SUCCESS (0x00) Server successfully started
NX_DHCPV6_ALREADY_STARTED
(0xE91) Server is running already
NX_DHCPV6_NO_ASSIGNABLE_ADDRESSES
(0xEA7) Server has no assignable addresses to lease
NX_DHCPV6_INVALID_GLOBAL_INDEX
(0xE97) Global address index not set
NX_DHCPV6_NO_SERVER_DUID
(0xE92) No Server DUID created
status(variable) ThreadX and NetX Duo error status
NX_PTR_ERROR (0x16) Invalid pointer input

Allowed From

Threads

Example

```
/* Start the DHCPv6 Server task.  */
status = nx_dhcpv6_server_start(&dhcp_server_0);

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

See Also

nx_dhcpv6_server_resume, nx_dhcpv6_server_suspend

nx_dhcpv6_retrieve_ip_address_lease

Get an IP address lease from the Server table

Prototype

```
UINT _nx_dhcpv6_retrieve_ip_address_lease(
          NX_DHCPV6_SERVER *dhcpv6_server_ptr, UINT table_index,
          NXD_ADDRESS *lease_IP_address, ULONG *T1, ULONG *T2,
          ULONG *valid_lifetime, ULONG *preferred_lifetime)
```

Description

This service retrieves an IP address lease record from the Server table at the specified table index location. This can be done before or after retrieving Client record data.

The capability of storing and retrieving data between the DHCPv6 Server and non volatile memory is a requirement of the DHCPv6 protocol. It makes no difference in what order IP lease data and Client record data is saved to nonvolatile memory.

Note: it is not recommended to copy data to or from Server tables without stopping or suspending the DHCPv6 Server first.

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server
table_index	Table index to store lease at
lease_IP_address	Pointer to IP address leased to theClient
T1	Client requested renew time
T2	Client requested rebind time
valid_lifetime	Client lease becomes deprecated
preferred_lifetime	Client lease becomes invalid

Return Values

NX_SUCCESS (0x00) Server successfully started
NX_DHCPV6_PARAMETER_ERROR
(0xE93) Invalid IP lease data input
NX_PTR_ERROR (0x16) Invalid pointer input

Allowed From

Application code

Example

```
/* Retrieve the DHCPv6 Server lease data.  */
For (I = 0; I < NX_DHCPV6_MAX_LEASES; i++)
{

    /* Get the next lease record. */
    status = nx_dhcpv6_server_startdhcpv6_server_ptr, i, &next_ipv6_address, &T1,
                   &T2, &preferred_lifetime, &valid_lifetime);

    /* The host application then saves this record to memory.
}

/* If status is NX_SUCCESS the Server data is successfully downloaded.  */
```

See Also

nx_dhcpv6_retrieve_client_record, nx_dhcpv6_add_ip_address_lease, nx_dhcpv6_add_client_record

nx_dhcpv6_add_ip_address_lease

Add an IP address lease to the Server table

Prototype

```
UINT _nx_dhcpv6_add_ip_address_lease(
      NX_DHCPV6_SERVER *dhcpv6_server_ptr, UINT table_index, NXD_ADDRESS
      *lease_IP_address, ULONG T1, ULONG T2,
      ULONG valid_lifetime, ULONG preferred_lifetime)
```

Description

This service loads IP lease data from a previous DHCPv6 Server session from non volatile memory to the Server lease table. This is not necessary if the Server is running for the first time and has no previous lease data. If this is the case the host application must create an IP address range for assigning IP addresses, using the *nx_dhcpv6_create_ip_address_range*service. The data is sufficient to reconstruct a DHCPv6 lease record. The table index need not be specified. If set to 0xFFFFFFFF (infinity) the DHCPv6 Server will find the next available slot to copy the data to.

Note: uploading IP lease data MUST be done before uploading Client records; both MUST be done before (re)starting the DHCPv6 Server.

The capability of storing and retrieving data between the DHCPv6 Server and non volatile memory is a requirement of the DHCPv6 protocol.

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server
table_index	Table index to store lease at
lease_IP_address	Pointer to IP address leased to theClient
T1	Client requested renew time
T2	Client requested rebind time
valid_lifetime	Client lease becomes deprecated
preferred_lifetime	Client lease becomes invalid

Return Values

NX_SUCCESS	(0x00)	Server successfully started
NX_DHCPV6_TABLE_FULL	(0xEC4)	No room for more lease data
NX_DHCPV6_INVALID_INTERFACE_IP_ADDRESS	(0xE95)	Lease data does not appear to be on link with Server DHCPv6 interface
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid IP lease data input

NX_PTR_ERROR (0x16) Invalid pointer input

Allowed From

Application code

Example

```
/* Copy the IP lease data to the Server address table.  Note that the table index is
   defaulted to 0xFFFFFFFF meaning the DHCPv6 Server will find an empty slot for each
   lease.  */

   For(I = 0; I < NX_DHCPV6_MAX_LEASES; i++)
   {
       status = nx_dhcpv6_add_ip_address_lease(dhcpv6_server_ptr, 0xFFFFFFFF,
               &next_ipv6_address, &T1, &T2, &preferred_lifetime, &valid_lifetime);

       /* Get the next lease address from memory… */
   }


   /* If status is NX_SUCCESS the lease data was successfully uploaded. It is opk
       to add the Client records to the Server table now.  */
```

See Also

nx_dhcpv6_retrieve_client_record, nx_dhcpv6_retrieve_ip_address_lease, nx_dhcpv6_add_client_record

nx_dhcpv6_add_client_record

Add a Client record to the Server table

Prototype

```
UINT _nx_dhcpv6_add_client_record(NX_DHCPV6_SERVER *dhcpv6_server_ptr,
                    UINT table_index, ULONG message_xid,
                    NXD_ADDRESS *client_address, UINT client_state,
                    ULONG IP_lease_time_accrued, ULONG valid_lifetime, UINT
                    duid_type,  UINTduid_hardware,
                    ULONG physical_address_msw,
                    ULONG physical_address_lsw, ULONG duid_time,
                    ULONG duid_vendor_number, UCHAR *duid_vendor_private, UINT
                    duid_private_length)
```

Description

This service copies Client data from non volatile memory to the Server table one record at a time. This is only necessary if the Server is being rebooted and has Client data from a previous session to restore from memory. If a Server has no previous data, the DHCPv6 Server will initialize the Client table to be able for adding Client records.

It is not necessary to specify the table index. If set to 0xFFFFFFFF (infinity) the DHCPv6 Server will locate the next available slot. The DHCPv6 Server can reconstruct a Client record from this data.

Note #1: the host application MUST upload the IP lease data BEFORE the Client record data. This is so that internally the DHCPv6 Server can cross link the tables so that each Client record is joined with its corresponding IP lease record in their respective tables. See *nx_dhcpv6_add_ip_address_lease* for details on uploading IP lease data from memory.

Note #2: depending on DUID type, not all data must be supplied. For example if a Client has a vendor assigned DUID type, it can send in zero for DUID Link Layer parameters (MAC address, hardware type, DUID time).

The capability of storing and retrieving data between the DHCPv6 Server and non volatile memory is a requirement of the DHCPv6 protocol.

Input Parameters

dhcpv6_server_ptr Pointer to DHCPv6 Server

Return Values

NX_SUCCESS (0x00) Server successfully started
NX_ INVALID_PARAMETERS
(0x4D) Invalid non pointer input
NX_DHCPV6_TABLE_FULL

(0xEC4) No empty slots left for adding another Client record

NX_DHCPV6_ADDRESS_NOT_FOUND
(0xEA8) Client assigned address not found in Server lease table.
NX_PTR_ERROR (0x16) Invalid pointer input

Allowed From

Application code

Example

```
/*Add the IP lease data and Client records back to the server before starting
theServer.  */

    /* Copy the 'lease data' to the server table FIRST. */
for (i = 0; i< NX_DHCPV6_MAX_LEASES; i++)
    {


        /* Add the next lease record. Let the server find the next
available slot. */
        status = nx_dhcpv6_add_ip_address_lease(dhcpv6_server_ptr,
                0xFFFFFFFF,,&next_ipv6_address, NX_DHCPV6_DEFAULT_T1_TIME,
                NX_DHCPV6_DEFAULT_T2_TIME, NX_DHCPV6_DEFAULT_PREFERRED_TIME,
                NX_DHCPV6_DEFAULT_VALID_TIME);

if (status != NX_SUCCESS)
return status;

        /* Get the next IP lease record from memory. */
        …
    }

    /* Copy the client records to the Server table NEXT.
for (i = 0; i< NX_DHCPV6_MAX_LEASES; i++)
    {


        /* Add the next client record. Let the server find the next
available slot. */
        status = nx_dhcpv6_add_client_record(dhcpv6_server_ptr, 0xFFFFFFFF,
                message_xid, &client_ipv6_address, NX_DHCPV6_STATE_BOUND,
                IP_lifetime_time_accrued, valid_lifetime, duid_type, duid_hardware,
                physical_address_msw, physical_address_lsw,
                duid_time, 0, NX_NULL, 0);

if (status != NX_SUCCESS)
return status;

        /* Get the next Client record from memory. */
        …
    }


/* If status is NX_SUCCESS the Server data was successfully restored and it is ok to
    start the DHCPv6 server now. */
```

See Also

nx_dhcpv6_retrieve_client_record, nx_dhcpv6_retrieve_ip_address_lease, nx_dhcpv6_add_ip_address_lease

nx_dhcpv6_retrieve_client_record

Retrieve a Client record from the Server table

Prototype

```
UINT _nx_dhcpv6_retrieve_client_record(
                NX_DHCPV6_SERVER *dhcpv6_server_ptr,
                UINT table_index, ULONG *message_xid,
                NXD_ADDRESS *client_address, UINT *client_state,
                ULONG IP_lease_time_accrued,
                ULONG *valid_lifetime, UINT *duid_type,
                UINT *duid_hardware, ULONG *physical_address_msw,
                ULONG *physical_address_lsw, ULONG *duid_time,
                ULONG *duid_vendor_number,
                UCHAR *duid_vendor_private,
                UINT *duid_private_length)
```

Description

This service copies the essential data from the Server's Client record table for storage to non-volatile memory. The Server can reconstruct an adequate Client record from such data in the reverse process (uploading data to the Server table). Regardless of the DUID type, none of the pointers can be NULL pointers; data is initialized to zero for all parameters. For example, if the Client DUID type is Link Layer Plus Time, the vendor number is returned as zero and the private ID is an empty string.

The capability of storing and retrieving data between the DHCPv6 Server and non volatile memory is a requirement of the DHCPv6 protocol. It makes no difference in what order IP lease data and Client record data is saved to nonvolatile memory.

Note: it is not recommended to copy data to or from Server tables without stopping or suspending the DHCPv6 Server first.

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server
table_index	Index into Server's client table
message_xid	Client Server Transaction ID
client_address	IPv6 address leased to Client
client_state	Client DHCPv6 state (e.g. bound)
IP_lease_time_accrued	Time expired on lease already
dhcpv6_server_ptr	Pointer to DHCPv6 Server
dhcpv6_server_ptr	Pointer to DHCPv6 Server

Return Values

NX_SUCCESS	(0x00)	Server successfully started
NX_DHCPV6_INVALID_DUID	(0xECC)	Invalid or inconsistent DUID data
NX_PTR_ERROR	(0x16)	Invalid pointer input

NX_INVALID_PARAMETERS
(0x4D) Invalid non pointer input

Allowed From

Application code

Example

```
/* Retrieve the Client records from the DHCPv6 Server table.  */
For (i = 0; i< NX_MAX_DHCPV6_CLIENTS; i++)
{
    status = nx_dhcpv6_retrieve_client_recorddhcpv6_server_ptr, i, &message_xid,
            &client_ipv6_address, &client_state, &IP_lifetime_time_accrued,
            valid_lifetime, &duid_type, &duid_hardware, &physical_address_msw,
            &physical_address_lsw, &duid_time, &duid_vendor_number, &private_id[0],
            &length);

    /* The host application can save this data to memory now.
}

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

See Also

nx_dhcpv6_add_client_record, nx_dhcpv6_retrieve_ip_address_lease, nx_dhcpv6_add_ip_address_lease

nx_dhcpv6_server_interface_set

Setthe interface index for Server DHCPv6 interface

Prototype

```
UINT _nx_dhcpv6_server_interface_set(
                  NX_DHCPV6_SERVER *dhcpv6_server_ptr,
                  UINT iface_index, UINT ga_address_index)
```

Description

This service sets the network interface on which the DHCPv6 Server handles DHCPv6 Client requests. Not that for versions of NetX Duo that do not support multihome, the interface value is defaulted to zero. The global address index is necessary to obtain the Server global address on its DHCPv6 interface. This is used by the DHCPv6 logic to ensure that lease addresses and other DHCPv6 data is on link with the DHCPv6 Server.

This must be called before the DHCPv6 server is started, even for applications on single homed devices or without multihome support.

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server
iface_index	Server DHCPv6 Server interface
ga_address_index	Index of Server global address in the Server IP instance address table

Return Values

NX_SUCCESS	(0x00)	Server successfully started
NX_INVALID_INTERFACE	(0x4C)	Interface does not exist
NX_NO_INTERFACE_ADDRESS	(0x50)	Global index exceeds the IP instance maximum IPv6 addresses (NX_MAX_IPV6_ADDRESSES)
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Application code

Example

```
/* Set the Server DHCPv6 interface to the primary interface. The global IP address is at
   the index 1 in the IP address table.  */
status = nx_dhcpv6_server_interface_set(&dhcp_server_0, 0, 1);

/* If status is NX_SUCCESS the Server interface is successfully set.  */
```

See Also

nx_dhcpv6_server_create, nx_dhcpv6_set_server_duid, nx_dhcpv6_server_start

nx_dhcpv6_set_server_duid

Set the Server DUID for DHCPv6 packets

Prototype

```
UINT _nx_dhcpv6_set_server_duid(NX_DHCPV6_SERVER *dhcpv6_server_ptr,
                    UINT duid_type, UINT hardware_type,
                    ULONG mac_address_msw, ULONG mac_address_lsw,
                    ULONG time)
```

Description

This service sets the Server DUID and must be called before the host application starts the Server. For link layer and link layer time DUID types, the host application must supply the hardware type and MAC address data. For link layer time DUIDs, the time pointer must point to a valid time. The number of seconds since Jan 1, 2000 is a typical seed value. If the Server DUID type is the enterprise, vendor assigned type, the DUID will be created from the user configurable options NX_DHCPV6_SERVER_DUID_VENDOR_PRIVATE_ID and NX_DHCPV6_SERVER_DUID_VENDOR_ASSIGNED_ID, and the time and MAC address values can be set to NULL.

Note: It is the host application's responsibility to save the Server DUID parameters to nonvolatile memory such that it uses the same DUID in messages to Clients between reboots. This is a requirement of the DHCPv6 protocol (RFC 3315).

Input Parameters

dhcpv6_server_ptr	Pointer to DHCPv6 Server
duid_type	DHCPv6 Server DUID type
hardware_type	Hardware type (e.g. Ethernet)
mac_address_msw	Pointer to DHCPv6 Server
mac_address_lsw	Pointer to DHCPv6 Server
time	Time value for DUID

Return Values

NX_SUCCESS	(0x00)	Server successfully suspended
NX_DHCPV6_INVALID_SERVER_DUID	(0XE98)	Unknown or unsupported DUID type
NX_INVALID_PARAMETERS	(0x4D)	Invalid non pointer input
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Application code

Example

```
/* Set the DHCPv6 ServerDUID as Link layer plus time, over Ethernet hardware.  */
    duid_time = SECONDS_SINCE_JAN_1_2000_MOD_32 + rand();

status = nx_dhcpv6_set_server_duid(&dhcp_server_0,1, 0x6,
                physical_address_msw,physical_address_lsw,duid_time);

/* If status is NX_SUCCESS the ServerDUID is successfully set.  */
```

See Also

nx_dhcpv6_process_duid, nx_dhcpv6_check_duids_same, nx_dhdpv6_add_duid

Appendix A – DHCPv6 Option Codes

Option	Code	Description
Client Identifier DUID	1	Uniquely identifies a Client host on the network
Server Identifier (DUID)	2	Uniquely identifies the DHCPv6Server host on the network
Identity Association for Non Temporary Addresses (IANA)	3	Parameters for a non temporary IP address assignment
Identity Association for Temporary Addresses (IATA)	4	Parameters for a temporary IP address assignment
IA Address	5	Actual IPv6 address and IPv6 address lifetimes to be assigned to the Client
Option Request	6	A list of information requests to obtain network information such as DNS server and other network configuration parameters.
Preference	7	Included in server Advertise message to client to influence the Client's choice of servers. The Client must choose a server with higher the preference value over other servers. 255 is the maximum value, while zero indicates the client can choose any server replying back to them
Elapsed Time	8	Contains the time (in 0.01 seconds) when the Client initiates the DHCPv6 exchange with the server. Used by secondary server(s) to determine if the primary server responds in time to the Client request.
Relay Message	9	Contains the original message in Relay message
Authentication	11	Contains information to authenticate the identity and content of DHCPv6 messages
Server Unicast	12	Server sends this option to let the Client know that the server will accept unicast messages directly from the Client instead of multicast.

IATA, Relay Message, Authentication and Server Unicast options are not supported in this release of NetX Duo DHCPv6 Server. The current DHCPv6 protocol option code 10 is left undefined in RFC 3315.

Appendix B - DHCPv6 Server Status Codes

Name	Code	Description
Success	0	Success
Unspecified Failure	1	Failure, reason unspecified; this status code is set by the Server to indicate a general failure in granting the Client request not matching the other codes
NoAddress Available	2	Server has no addresses available to assign to the Client
NoBinding	3	Client IA address (binding) is not available e.g. the requested IP address is not available for the Server to lease or assigned to another Client.
NotOnLink	4	The prefix for the address indicates the IP address is not an on link address
UseMulticast	5	Sent by a Server in response to receiving a Client message using the Server's unicast address instead of the *All_DHCP_Relay_Agents_and_Servers* multicast address

Appendix C - DHCPv6 Unique Identifiers (DUIDs)

DUID Type	Code	Description
DUID-LLT	1	Link layer plus time; identifier based on link layer address and time
DUID-EN	2	Enterprise; Assigned by Vendor Based on Enterprise Number
DUID-LL	3	Link layer; Based on Link-layer Address only

Appendix D Advanced DHCPv6 Server Example

This is an advanced DHCPv6 Server which demonstrates saving and retrieving the Server's IP address lease table and Client record tables from non volatile memory, as required by the RFC 3315.

In this example, the include file *nxd_dhcpv6_server.h* is brought in at line 7. Next, the NetX Duo DHCPv6 Server application thread is created at line 81 in the example code below. Note that the DHCPv6 control block "*dhcp_server_0*" was defined as a global variable at line 19 previously.

Before creating the NetX Duo DHCPv6 Server instance, the demo creates packet pool for sending DHCPv6 messages in line 84, creates an IP thread interface in line 102, and enables UDP in NetX Duo in line 116.

The successful creation of NetX Duo DHCPv6 Server in line 136 includes the the two optional callbacks function described in Chapter 1.It enables IPv6 and ICMPv6 necessary for NetX Duo to process IPv6 and DHCPv6 operations in line 162-163. Before the DHCPv6 Server thread is ready to run, the DHCPv6 server must validate its IPv6 address(167-180),and define its DHCPv6 interface in lines 208-209. The *nx_dhcpv6_set_server_duid* service is called to create the Server if no Server DUID has been previously created in line 266. The Server sets up an IP address range for creating a list of assignable addresses. If data is saved from a previous session, it retrieves Client records and IPv6 lease data from memory in lines 283-318. It also creates its Server DUID, or if one was previously created, retrieves the DUID data from user specified storage. This is necessary to reproduce a consistent Server DUID across reboots. Optionally the host application defines a DNS server for Clients requesting DNS server configuration.

Next, the host starts the DHCPv6Server in line 329. This creates the DHCPv6 Server UPD socket and activates NetX Duo DHCPv6 Server timers. Then the Server waits to receive Client requests. While it can service many Clients it can only process a single Client request at a time.

The remainder of the example contains host implementations for saving and retrieving Server tables of its assignable IPv6 address pool and Client records to and from non volatile memory respectively. It also contains an option handler for options requested by DHCPv6 Clients that are not supported directly by the NetX Duo DHCPv6 Server (only the DNS server option is currently supported). Lastly there is a code for demonstrating how to save and retrieve 'non volatile time' by which the Server keeps track of assigned IP lease expiration.

```
1     /* This is a small demo of the NetX Duo DHCPv6 Server for the high-performance
2        NetX Duo TCP/IP stack.  */
3
4     #include    <stdio.h>
5     #include   "tx_api.h"
```

```
#include   "nx_api.h"
#include   "nxd_dhcpv6_server.h"


#define     DEMO_STACK_SIZE         2048



/* Define the ThreadX and NetX Duo object control blocks...  */

TX_THREAD               thread_0;
NX_PACKET_POOL          pool_0;
NX_IP                   ip_0;
NX_DHCPV6_SERVER        dhcp_server_0;


/* Define the counters used in the demo application...  */

ULONG                   thread_0_counter;
ULONG                   state_changes;
ULONG                   error_counter;

#define                 SERVER_PRIMARY_ADDRESS   IP_ADDRESS(192,2,2,66)

/* Define thread prototypes.  */

void    thread_0_entry(ULONG thread_input);

/***** Substitute your ethernet driver entry function here *********/
void    nx_etherDriver_mcf5485(struct NX_IP_DRIVER_STRUCT *driver_req);


/* Define some DHCPv6 parameters. */

#define DHCPV6_IANA_ID  0xC0DEDBAD
#define DHCPV6_T1       NX_DHCPV6_INFINITE_LEASE
#define DHCPV6_T2       NX_DHCPV6_INFINITE_LEASE


/* Declare NetX Duo DHCPv6 Server callbacks. */

VOID dhcpv6_decline_handler(struct NX_DHCPV6_SERVER_STRUCT *dhcpv6_server_ptr,
                                NX_DHCPV6_CLIENT *dhcpv6_client_ptr, UINT message_type);
VOID dhcpv6_option_request_handler(NX_DHCPV6_SERVER *dhcpv6_server_ptr, UINT option_request,
                                UCHAR *buffer_ptr, UINT *index);

/* Declare helper functions for the DHCPv6 Server host application. */
VOID dhcpv6_get_time_handler(ULONG *realtime);
UINT dhcpv6_create_ip_address_range(NX_DHCPV6_SERVER *dhcpv6_server_ptr, UINT
                                *addresses_added);
UINT dhcpv6_restore_ip_lease_table(NX_DHCPV6_SERVER *dhcpv6_server_ptr);
UINT dhcpv6_restore_client_records(NX_DHCPV6_SERVER *dhcpv6_server_ptr);
UINT dhcpv6_retrieve_ip_address_lease(NX_DHCPV6_SERVER *dhcpv6_server_ptr);
UINT dhcpv6_retrieve_client_records(NX_DHCPV6_SERVER *dhcpv6_server_ptr);


/* Define main entry point.  */

intmain()
{

    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();
}


/* Define what the initial system looks like.  */

void    tx_application_define(void *first_unused_memory)
{

CHAR    *pointer;
UINT    status;

    /* Setup the working pointer.  */
    pointer =  (CHAR *) first_unused_memory;

    /* Create the main thread.  */
    tx_thread_create(&thread_0, "thread 0", thread_0_entry, 0,
            pointer, DEMO_STACK_SIZE,
            1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);

    pointer =  pointer + DEMO_STACK_SIZE;

```

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

    /* Create a packet pool.  */
    status =  nx_packet_pool_create(&pool_0, "NetX Main Packet Pool", NX_DHCPV6_PACKET_SIZE,
                                    pointer, NX_DHCPV6_PACKET_POOL_SIZE);
                                    pointer = pointer + NX_DHCPV6_PACKET_POOL_SIZE;

    /* Check for pool creation error.  */
    if (status != NX_SUCCESS)
    {
        error_counter++ ;
        return;
    }

    /* Create an IP instance.  */
    status = nx_ip_create(&ip_0, "NetX IP Instance 0", SERVER_PRIMARY_ADDRESS,
                          0xFFFFFF00UL, &pool_0, nx_etherDriver_mcf5485,
                          pointer, 2048, 1);

    pointer =  pointer + 2048;

    /* Check for IP create errors.  */
    if (status != NX_SUCCESS)
    {
        error_counter++ ;
        return;
    }

    /* Enable UDP traffic for sending DHCPv6 messages.  */
    status =  nx_udp_enable(&ip_0);

    /* Check for UDP enable errors.  */
    if (status != NX_SUCCESS)
    {
        error_counter++;
        return;
    }

    /* Enable ICMP.  */
    status =  nx_icmp_enable(&ip_0);

    /* Check for ICMP enable errors.  */
    if (status != NX_SUCCESS)
    {
        error_counter++ ;
        return;
    }

    /* Create the DHCPv6 Server. */
    status =  nx_dhcpv6_server_create(&dhcp_server_0, &ip_0, "DHCPv6 Server", &pool_0,
                         pointer, 2048, dhcpv6_decline_handler,
                 dhcpv6_option_request_handler);

    /* Check for errors.  */
    if (status != NX_SUCCESS)
    {
        error_counter++;
    }

    /* Yield control to DHCPv6 threads and ThreadX. */
    return;
}


/* Define the Server host application thread. */

void    thread_0_entry(ULONG thread_input)
{

UINT        status;
NXD_ADDRESS ipv6_address_primary, dns_ipv6_address;
ULONGduid_time;
UINTiface_index, ga_address_index;
UINTaddresses_added;


    /* Make the DHCPv6 Server IPv6 and ICMPv6 enabled. */
    nxd_ipv6_enable(&ip_0);
    nxd_icmp_enable(&ip_0);

    memset(&ipv6_address_primary,0x0, sizeof(NXD_ADDRESS));

    ipv6_address_primary.nxd_ip_version = NX_IP_VERSION_V6 ;
```

```
    ipv6_address_primary.nxd_ip_address.v6[0] = 0x20010db8;
    ipv6_address_primary.nxd_ip_address.v6[1] = 0xf101;
    ipv6_address_primary.nxd_ip_address.v6[2] = 0x00000000;
    ipv6_address_primary.nxd_ip_address.v6[3] = 0x00000101;




    /* Wait till the IP task thread has had a chance to set the device MAC address. */

    tx_thread_sleep(10);





/* Set the primary interface link local address (address index 0). This
    will use the host MAC address to build the link local address. */

    nxd_ipv6_linklocal_address_set(&ip_0, NULL);

/* Set the single homed host global IP address. */

    nxd_ipv6_global_address_set(&ip_0, &ipv6_address_primary, 64);


    tx_thread_sleep(500);


    /* Set the server interface for DHCP communications. */
    iface_index = 0;
    ga_address_index = 1;

    /* Set the DHCPv6 server interface to the primary interface and global address index. */
    status = nx_dhcpv6_server_interface_set(&dhcp_server_0, iface_index, ga_address_index);

    /* Wait for DHCP to assign the IP address.  */
    do
    {

        /* Check for address resolution.  */
        status =  nx_ip_status_check(&ip_0, NX_IP_ADDRESS_RESOLVED, (ULONG *)
                                     &status, 100000);

        /* Check status.  */
        if (status)
        {

            tx_thread_sleep(20);
        }

    } while (status != NX_SUCCESS);

    dns_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    dns_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
    dns_ipv6_address.nxd_ip_address.v6[1] = 0x0000f101;
    dns_ipv6_address.nxd_ip_address.v6[2] = 0x00000000;
    dns_ipv6_address.nxd_ip_address.v6[3] = 0x00000107;

    status = nx_dhcpv6_create_dns_address(&dhcp_server_0, &dns_ipv6_address);

    /* Check for errors. */
    if (status != NX_SUCCESS)
    {

        error_counter++;
        return;
    }

    /* Set the server DUID before starting the DHCPv6 server. You will also need to set the
       Server DUID if you are restoring Client data from non volatile memory.

       This demo will create a server DUID of the link layer time DUID type.

       Note #1: The RFC 3315 Sect 9.2 recommends link layer time DUID type over link layer
       DUID type to minimize the chances of 'collisions' or identical DUIDs between hosts,
        particularly clients.

       Note #2: If the client or server host is rebooting, RFC 3315 Sect 9.2 requires the
       host retrieve its previously created DUID data rather than create one from new data.

       For a Link layer time DUID, retrieve a time value.  If the DHCPv6 server has not
       created a server DUID previously, this function should provide a new value; otherwise
        this function must retrieve the time data used in the previously created server
```

```
          DUID. For link layer and enterprise type DUIDs, the 'duid_time' data is not
          necessary. */
          dhcpv6_get_time_handler(&duid_time);


       /* For DUID types that do not require time, the 'duid_time' input can be left at zero.
           The DUID_TYPE and HW_TYPE are configurable options that are user defined in
     nxd_dhcpv6_server.h.  */


      status = nx_dhcpv6_set_server_duid(&dhcp_server_0,
                                      NX_DHCPV6_SERVER_DUID_TYPE, NX_DHCPV6_SERVER_HW_TYPE,
                                      dhcp_server_0.nx_dhcpv6_ip_ptr ->
                                       nx_ip_arp_physical_address_msw,
                                      dhcp_server_0.nx_dhcpv6_ip_ptr ->
                                       nx_ip_arp_physical_address_lsw,
                                      duid_time);
      if (status != NX_SUCCESS)
      {
          error_counter++ ;
          return;
      }


      /* The next step is to set up the server IP lease and Client record tables.  If no
         previous data exists, the host application only needs to create an IP address range
         of assignable IP addresses, and set the size of the tables, NX_DHCPV6_MAX_CLIENTS
         and NX_DHCPV6_MAX_LEASES in nxd_dhcpv6_server.h.  */

#ifndef RESTORE_SERVER_DATA

      /* Create the ip address table on the primary server network interface. */
      status = dhcpv6_create_ip_address_range(&dhcp_server_0, &addresses_added);

      if (status != NX_SUCCESS)
      {

          error_counter++;
          return;
      }

#else

      /* RFC 3315 requires that DHCPv6 servers be able to store and retrieve lease data to and
         from non-volatile memory so that DHCPv6 server may remain uninterrupted across server
          reboots. */
      status = dhcpv6_restore_ip_lease_table(&dhcp_server_0);

      if (status != NX_SUCCESS)
      {

          error_counter++;
          return;
      }

      status = dhcpv6_restore_client_records(&dhcp_server_0);

      if (status != NX_SUCCESS)
      {

          error_counter++;
          return;
      }


#endif /* RESTORE_SERVER_DATA */

      /*Check for error. */
      if (status != NX_SUCCESS)
      {

          error_counter++;
          return;
      }

      /* Start the NetX Duo DHCPv6 server!  */
      status =  nx_dhcpv6_server_start(&dhcp_server_0);

      /* Check for errors. */
      if (status != NX_SUCCESS)
      {
          error_counter++;
      }

```

```
    return;
}

/* Simulate a handler with access to a real time clock and non volatile memory storage. This
   service is required for a link layer time DUID to create a time value as part of
   the DUID.  A default value is provided below. The time value serves
   no actual function, but increases the chances of a unique host DUID.

   It is the host's responsibility to save the 'time' data created for the server DUID to
   memory.  The DHCPv6 server should always use a previously created its server DUID as per
   RFC 3315 Sect. 9.2. */
VOID dhcpv6_get_time_handler(ULONG *realtime)
{


    /* Check if the DHCPv6 server has previously created a DUID. If so
       return this time value to the host application. */
    /*********** insert your application logic here **************/

    /* Otherwise create time data.  One can use a random number incremented
       to the number of seconds since JAN 1, 2000 to
       create a unique time value.  */
    *realtime = SECONDS_SINCE_JAN_1_2000_MOD_32;

    return;
}


/* Create an IP address lease table based on from a range of available addresses. */
UINT dhcpv6_create_ip_address_range(NX_DHCPV6_SERVER *dhcpv6_server_ptr,
                                        *UINT *addresses_added)
{

UINT status;
NXD_ADDRESS start_ipv6_address;
NXD_ADDRESS end_ipv6_address;


    start_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    start_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
    start_ipv6_address.nxd_ip_address.v6[1] = 0x00000f101;
    start_ipv6_address.nxd_ip_address.v6[2] = 0x0;
    start_ipv6_address.nxd_ip_address.v6[3] = 0x00000110;

    end_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
    end_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
    end_ipv6_address.nxd_ip_address.v6[1] = 0x0000f101;
    end_ipv6_address.nxd_ip_address.v6[2] = 0x00000000;
    end_ipv6_address.nxd_ip_address.v6[3] = 0x00000120;

    status = nx_dhcpv6_create_ip_address_range(dhcpv6_server_ptr, &start_ipv6_address,
                                               &end_ipv6_address, addresses_added);

    return status;

}

/* Demonstrate how to use NetX Duo DHCPv6 Server API to upload data from memory
  to the DHCPv6 server's IP lease tables. */
UINT dhcpv6_restore_ip_lease_table(NX_DHCPV6_SERVER *dhcpv6_server_ptr)
{

NXD_ADDRESS     next_ipv6_address;
UINTi;
UINT            status;


    /* Set the starting IP address. */
    next_ipv6_address.nxd_ip_version = 6;
    next_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
    next_ipv6_address.nxd_ip_address.v6[1] = 0x00000f101;
    next_ipv6_address.nxd_ip_address.v6[2] = 0x0;
    next_ipv6_address.nxd_ip_address.v6[3] = 0x00000110;


    /* Copy the 'lease data' to the server table. */
    for (i = 0; i< NX_DHCPV6_MAX_LEASES; i++)
    {

        /* These are assigned address leases. */

        status = nx_dhcpv6_add_ip_address_lease(dhcpv6_server_ptr, i, &next_ipv6_address,
                  NX_DHCPV6_DEFAULT_T1_TIME, NX_DHCPV6_DEFAULT_T2_TIME,
                   X_DHCPV6_DEFAULT_PREFERRED_TIME, NX_DHCPV6_DEFAULT_VALID_TIME);
```

```

        if (status != NX_SUCCESS)
            return status;

        /* Simulate the next IP address in the table. */
        next_ipv6_address.nxd_ip_address.v6[3]++;
    }

    return NX_SUCCESS;
}

/* Demonstrate how to use NetX Duo DHCPv6 Server API to download data to local memory and
   eventually nonvolatile memory from the DHCPv6 server's IP lease tables. This might be
   called after the a certain duration of operation and after stopping Server task (e.g.
   before rebooting or for making a backup).*/

UINT  dhcpv6_retrieve_ip_address_lease(NX_DHCPV6_SERVER *dhcpv6_server_ptr)
{

NXD_ADDRESS     next_ipv6_address;
ULONG           T1, T2, valid_lifetime, preferred_lifetime;
UINTi;
UINT            status;





    for (i = 0; i< NX_DHCPV6_MAX_LEASES; i++)
    {

        T1 = 0;
        T2 = 0;
        valid_lifetime = 0;
        preferred_lifetime = 0;
        memset(&next_ipv6_address, 0, sizeof(NXD_ADDRESS));

        /* Get the next lease from the table. */
        status = nx_dhcpv6_retrieve_ip_address_lease(dhcpv6_server_ptr, i,
                 &next_ipv6_address, &T1, &T2, &preferred_lifetime, &valid_lifetime);

        if (status != NX_SUCCESS)
            return status;

        /* At this point the host application would store this record to NV memory. */
    }

    return NX_SUCCESS;
}

/* Demonstrate how to use NetX Duo DHCPv6 Server API to upload data from memory
  to the DHCPv6 server's client record tables. */
UINT dhcpv6_restore_client_records(NX_DHCPV6_SERVER *dhcpv6_server_ptr)
{

UINTi;
UINT                status;

/* Create data to simulate client records stored in memory. */
NXD_ADDRESS         client_ipv6_address;
UINTduid_type = 1;
UINTduid_hardware = NX_DHCPV6_HW_TYPE_IEEE_802;
ULONGmessage_xid = 0xabcd;
UINTduid_time = 0x1234567;
ULONGphysical_address_msw = 0x01;
ULONGphysical_address_lsw = 0x02030405;
ULONGIP_lifetime_time_accrued = 200000;  /* lease time accrued (ticks) */
ULONGvalid_lifetime = 300000; /* expiration on the lease (ticks) */
ULONGenterprise_number = 0xaaaa;
UCHARprivate_id[8];
UINT                length;


    /* Set the Client IP address. */
    client_ipv6_address.nxd_ip_version = 6;
    client_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
    client_ipv6_address.nxd_ip_address.v6[1] = 0x00000f101;
    client_ipv6_address.nxd_ip_address.v6[2] = 0x0;
    client_ipv6_address.nxd_ip_address.v6[3] = 0x00000110;


    /* Copy the 'lease data' to the server table. */
    for (i = 0; i< 10; i++)
    {
```

```
        /* Simulate a Client record with a vendor assigned DUID. */
        if (i == 0)
        {
             duid_type = NX_DHCPV6_SERVER_DUID_TYPE_VENDOR_ASSIGNED;

             memcpy(&private_id[0], "Corp_XYZ", sizeof("Corp_XYZ"));
             length = sizeof("Corp_XYZ") + 4;
             status = nx_dhcpv6_add_client_record(dhcpv6_server_ptr, i, message_xid,
                  &client_ipv6_address, NX_DHCPV6_STATE_BOUND, IP_lifetime_time_accrued,
                  valid_lifetime, duid_type, duid_hardware, physical_address_msw,
                  physical_address_lsw, duid_time, enterprise_number,
                  &private_id[0], length);
        }
        /* Simluate client record with a link layer DUID.  */
        else
        {
               status = nx_dhcpv6_add_client_record(dhcpv6_server_ptr, i, message_xid,
                  &client_ipv6_address, NX_DHCPV6_STATE_BOUND, IP_lifetime_time_accrued,
       valid_lifetime, duid_type, duid_hardware, physical_address_msw,
                  physical_address_lsw, duid_time, 0, NX_NULL, 0);
        }

        /* Check for error. */
        if (status != NX_SUCCESS)
        {

            /* Check if the Client address is found in the IP lease table. */
            if (status == NX_DHCPV6_ADDRESS_NOT_FOUND)
            {

                /* It is not. Client state should be set to unbound/init.*/
            }
            else
            {

                /* Either the table is full or the index exceeds the bounds of the table. */
                return status;
            }
        }

        /* Simulate the Client IP address in the table. Leave all other client 'data' the
          same for the next record we'll 'restore'. */
        client_ipv6_address.nxd_ip_address.v6[3]++;
        physical_address_lsw++;
        message_xid++;
    }

    return NX_SUCCESS;
}

/* Demonstrate how to use NetX Duo DHCPv6 Server API to download data to local memory and
   eventually nonvolatile memory from the DHCPv6 server's client record tables. */

UINT dhcpv6_retrieve_client_records(NX_DHCPV6_SERVER *dhcpv6_server_ptr)
{

UINTi;
UINT                status;
NXD_ADDRESS         client_ipv6_address;
UINTduid_type;
UINTduid_hardware;
ULONGmessage_xid;
ULONGduid_time;
ULONGphysical_address_msw;
ULONGphysical_address_lsw;
ULONGIP_lifetime_time_accrued;  /* lease time accrued (ticks) */
ULONGvalid_lifetime; /* expiration on the lease (ticks) */
ULONGduid_vendor_number;
UCHARprivate_id[8];
UINT                length;
UINTclient_state;


    for (i = 0; i< 100; i++)
    {

        memset(&client_ipv6_address, 0,sizeof(NXD_ADDRESS));

        status = nx_dhcpv6_retrieve_client_record(dhcpv6_server_ptr, i, &message_xid,
          &client_ipv6_address, &client_state, &IP_lifetime_time_accrued,
          &valid_lifetime, &duid_type, &duid_hardware, &physical_address_msw,
          &physical_address_lsw, &duid_time, &duid_vendor_number, &private_id[0], &length);

        if (status != NX_SUCCESS)
```

```
        {
            /* The host application should handle error status returns depending on
               the specific error code.  See the user guide for error returns for
               this service. */
        }

    }

    return NX_SUCCESS;
}

/*  This is an optional callback for NetX DHCPv6 server to notify the host application
    that it has received either a DECLINE or RELEASE address from a Client. */

VOID dhcpv6_decline_handler(NX_DHCPV6_SERVER *dhcpv6_server_ptr, NX_DHCPV6_CLIENT
          *dhcpv6_client_ptr, UINT message_type)
{

    switch (message_type)
    {
        case NX_DHCPV6_MESSAGE_TYPE_DECLINE:


            /* Host application handles a declined address. The Server will
               mark the address as assigned elsewhere. Any other processing
               is left to the host application. */

        break;

        case NX_DHCPV6_MESSAGE_TYPE_RELEASE:

            /* Host application handles a released address. The Server will
               mark the released IP address as available for lease to other
               clients. Any other processing is left to the host application. */

        break;

        default:

            /* Unhandled message type */
            error_counter++;
        break;
    }

    return;
}

/*  This is an optional DHCPv6 server callback to handle client option request options. */
VOID dhcpv6_option_request_handler(NX_DHCPV6_SERVER *dhcpv6_server_ptr, UINT
                                 option_request, UCHAR *buffer_ptr, UINT *index)
{

UCHARoption_length = 10;
UCHARoption_code = 24;
ULONGmessage_word;


    if (option_request == 24)
    {

        message_word = option_code<< 16;
        message_word |= option_length;

        /* Adjust for endianness. */
        NX_CHANGE_ULONG_ENDIAN(message_word);

        /* Copy the option request option header to the packet. */
        memcpy(buffer_ptr + *index, &message_word, sizeof(ULONG));
        *index += sizeof(ULONG);

        /* Copy the code for domain search list. */
        *(buffer_ptr + *index) = 0x04;
        (*index)++;

        /* Adjusting for endianness is an exercise left for the reader. */
        memcpy(buffer_ptr + *index, "abc.com", sizeof("abc.com"));
        (*index) += sizeof("abc.com");
    }
    /* else unknown option; just return; no need to adjust buffer pointers.  */

    return;
}
```

Figure 6. Advanced NetX Duo DHCPv6 Server Application