Microsoft

Azure RTOS
NetX Duo DHCPv6 Client
User Guide

Published: February 2020

For the latest information, please see
azure.com/rtos

Part Number: 000-1050

Revision 6.0

Contents

Chapter 1
Introduction to DHCPv6 Client

In IPv6 networks, DHCPv6 replaces DHCP for dynamic global IP address assignment from a DHCPv6 Server, and offers most of the same features as well as many enhancements. This document will explain in detail how the NetX Duo DHCPv6 Client API is used to obtain IPv6 addresses.

DHCPv6 Communication

The DHCPv6 protocol uses UDP. The Client uses port 546 and the Server uses port 547 to exchange DHCPv6 messages. The Client uses its link local address for a source address to initiate the DHCPv6 requests to the DHCPv6 server(s) available. When the Client sends messages intended for all DHCPv6 servers on the network it uses the *All_DHCP_Relay_Agents_and_Servers* multicast address *FF02::01:02*. This is a reserved, link-scoped multicast address.

DHCPv6 Process of Requesting an IPv6 Address

To begin the process of requesting a global IPv6 address assignment, a Client first sends a SOLICIT message using the *nx_dhcpv6_send_solicit* service:

```
UINT  nx_dhcpv6_request_solicit(NX_DHCPV6 *dhcpv6_ptr)
```

This message is sent to all servers using the *All_DHCP_Relay_Agents_and_Servers* address. In the SOLICIT request, the Client may request the assignment of specific IPv6 address(es) as a hint to the Server. It can also request other network configuration information from the Server such as DNS server, NTP server and other options in the Option Request in the SOLICIT message.

A DHCPv6 Server that can service a Client request responds with an ADVERTISE message containing the IPv6 address(es) it can assign to the Client, the IPv6 address lease time and any additional information requested by the Client. The DHCPv6 Client protocol requires the Client to wait for a period of time to receive ADVERTISE messages from all DHCPv6 Servers on the network. It pre-processes each ADVERTISE message to be a valid message, and scans the option data for various DHCPv6 parameters. It also checks the Preference value in the Preference Option, if supplied by the Server. If more than one ADVERTISE message is received, the NetX DHCPv6 Client chooses the Server with the highest preference value received by the end of the wait period.

The exception is if the Client receives an ADVERTISE message with a preference value of 255. It accepts that message immediately and discards all subsequent ADVERTISE messages.

The wait period is defined as the retransmission period that the DHCPv6 Client waits before sending another SOLICIT message if it has not received a response from any Server. The initial retransmission timeout in the SOLICIT state is defined by to the DHCPv6 protocol described in RFC 3315 to be 1 second. Subsequent retransmission intervals, if the DHCPv6 Client fails to receive a valid Server response, are doubled up to a maximum of 120 seconds.

Having chosen the Server, the Client extracts data from the ADVERTISE message and sends a REQUEST message back to the Server to accept the assigned address information and lease times. The Server responds with a REPLY message to confirm the Client IPv6 address(es) are registered with the Server as assigned to the Client.

The DHCPv6 Client registers the assigned IPv6 address(es) with the IP instance (e.g NetX Duo). If configured for the Duplicate Address Detection (DAD) protocol (enabled by default), NetX Duo will automatically send Neighbor Solicit messages to verify the assigned address(es) are unique on the network. If so, it notifies the DHCPv6 Client when the each assigned address has been promoted from TENTATIVE to VALID. The DHCPv6 Client is promoted to the BOUND state and the device may use that IPv6 address to send and transmit messages. If the DAD protocol fails, NetX Duo notifies the DHCPv6 Client and the DHCPv6 Client sends a DECLINE message for the IPv6 address(es) assigned back to the Server and resets the DHCPv6 Client state to the INIT state.

Notification of Successful Address Assignment and Validation

The application can determine the result of the DHCPv6 Client address solicitation from the state changes if the DHCPv6 Client is configured with the state change callback in the *nx_dhcpv6_client_create* service. If it receives no response from the Server the state change observed is from SOLICIT to INIT. If it received a response from the Server but the Server is unable to assign the address, the application will be notified by the DHCPv6 Client server error callback if configured with one (also in *nx_dhcpv6_client_create).* If the Client achieved the BOUND state but then failed the DAD check, it will see a state change from BOUND to INIT. Note that an application that is enabled for DAD must allow time for the DAD check after starting the request process. Typically this is about 400-500 ticks (4-5 seconds in most cases).

Relinquishing an IPv6 Address

If and when the Client needs to release an assigned IPv6 address, it informs the DHCPv6 server by calling the *nx_dhcpv6_request_release* service:

```
UINT  nx_dhcpv6_request_release(NX_DHCPV6 *dhcpv6_ptr)
```

The DHCPv6 Client sends a unicast RELEASE message containing the assigned addresses to the Server who assigned the address and waits for a REPLY confirming the Server received the message.

Note: There is a different process for the Client that is powering down but plans to continue using the assigned IPv6 addresses on reboot. It does not send the RELEASE message on powering down unless it plans to request a new address on power up. See “Non Volatile Memory Requirements” for explanation of this situation.

DHCPv6 Lease Timeouts

The IPv6 lease assigned by the Server contains two timeout parameters, T1 and T2 in each Identity Association – Non Temporary Addresses (IANA) block. An IANA is described in elsewhere in this User Guide. If the elapsed time from when the DHCPv6 Client was bound to the assigned IPv6 address equals T1, the DHCPv6 Client automatically starts renewing the IPv6 address by sending a RENEW message. If the elapsed time equals T2, DHCPv6 Client automatically sends a REBIND message if it received no responses to its RENEW requests.

Two other IPv6 lease parameters, preferred and valid lifetime, are assigned with each Identity Association (IA) block contained in the IANA block. The preferred and valid lifetimes are when the assigned IPv6 address is deprecated or invalid, respectively. T1 must be less than the preferred lifetime. T2 must be less than the valid lifetime.

IP Thread Task Requirements

The NetX Duo DHCPv6 Client requires creation of a NetX Duo IP instance previous to creating the DHCPv6 Client to use DHCPv6 Client services.

UDP, IPv6, and ICMPv6 must be enabled on the IP instance prior to the using DHCPv6 Client.

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

Packet Pool Requirements

NetX Duo DHCPv6 Client creation requires a previously created 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 application will be sending.

A typical DHCPv6 message is about 200 bytes depending on the number of IA addresses and DHCPv6 options requested by the Client.

Network Requirements

The NetX Duo DHCPv6 Client requires the creation of UDP socket bound to port 546. The socket is created when the DHCPv6 Client task is created.

Non Volatile Memory Requirements

If a DHCPv6 Client releases its IPv6 lease with the DHCPv6 Server when powering down, and requests new IPv6 address(es) on reboot, then non volatile memory storage is not required. If a Client wishes to continue using its assigned lease, it must store certain information about the DHCPv6 Client to non volatile memory across reboots.

Non volatile memory requirements and the DHCPv6 Client API are discussed further in **Using the NetX Duo DHCPv6 Client** in Chapter Two.

NetX Duo DHCPv6 Client Limitations

The current release of the NetX Duo DHCPv6 Client has the following limitations:

- NetX Duo DHCPv6 Client does not support the Server Unicast option for sending unicast DHCPv6 messages to the DHCPv6 Server even if the Server indicates this is permitted.

- NetX Duo DHCPv6 Client does not support the Reconfigure request in which a Server initiates IPv6 address changes to the Clients on the network.

- NetX Duo DHCPv6 Client does not support the Enterprise format for the DHCPv6 Unique Identifier control block. It only supports Link Layer and Link Layer Plus Time format.

- NetX Duo DHCPv6 Client does not support Temporary Association (TA) address requests, but does support Non Temporary (IANA) option requests.

Multihome and Multiple Address Support

The DHCPv6 Client supports multiple interfaces and multiple addresses per interface. The DHCPv6 Client service, *nx_dhcpv6_client_set_interface* enables the Client application to set the network interface on which the application will be communicating with the DHCPv6 Server. The DHCPv6 Client defaults to the primary interface (index zero).

For multiple addresses per interface, the DHCPv6 Client keeps an internal list of addresses starting at index 0. Note that the same address registered with the DHCPv6 Client may not necessarily be located at the same index in the IP table of interface addresses.

For DHCPv6 Client services that retrieve information about the Client IPv6 address lease, some require an address index to be specified. An example for obtaining the preferred and valid lifetimes is shown below:

```
UINT nx_dhcpv6_get_valid_ip_address_lease_time(NX_DHCPV6 *dhcpv6_ptr,
                                               UINT address_index,
                                               NXD_ADDRESS *ip_address,
                                               ULONG preferred_lifetime,
                                               ULONG *valid_lifetime)
```

The Client application can also retrieve the number of valid IPv6 addresses assigned from the *nx_dhcpv6_get_valid_ip_address_count* service:

```
UINT nx_dhcpv6_get_valid_ip_address_count(NX_DHCPV6 *dhcpv6_ptr,
                                          UINT *address_count)
```

Legacy DHCPv6 Client services which were created before multiple addresses were supported in NetX Duo do not take an address index. Therefore, with these services, the data requested is taken from the primary global IA address, regardless how many IA addresses are assigned to the Client. An example is shown below:

```
UINT nx_dhcpv6_get_IP_address(NX_DHCPV6 *dhcpv6_ptr,
                              NXD_ADDRESS *ip_address)
```

NetX Duo DHCPv6 Client Callback Functions

nx_dhcpv6_state_change_callback

When the DHCPv6 Client changes to a new DHCPv6 state as a result of processing a DHCPv6 request, it notifies the application with this callback function.

nx_dhcpv6_server_error_handler

When the DHCPv6 Client receives a Server reply containing a *Status* option with a non-zero (non successful) status, it notifies the application with this callback which includes the Server error status code.

Note: Since these callback functions are called from the DHCPv6 Client thread task, the Client application must NOT call any NetX Duo DHCPv6 Client services that require mutex control of the DHCPv6 Client such as *nx_dhcpv6_start, nx_dhcpv6_stop,* and any of the API that send messages directly from the callback e.g. *nx_dhcpv6_request_release*.

DHCPv6 RFCs

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

Chapter 2
Installation and Use of the DHCPv6 Client

This chapter contains a description of various issues related to installation, setup, and usage of the NetX Duo DHCPv6 Client component.

Product Distribution

NetX Duo DHCPv6 Client 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_client.h	Header file for NetX DuoDHCPv6 Client
nxd_dhcpv6_client.c	Source code file for NetX Duo DHCPv6 Client
demo_netxduo_dhcpv6_client.c	Sample program demonstrating the setup of the NetX Duo DHCPv6 Client
nxd_dhcpv6_client.pdf	PDF description of NetX Duo DHCPv6 Client

NetX Duo DHCPv6 Client Installation

To use NetX Duo DHCPv6 Client API, the entire distribution mentioned above can 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_client.h* and *nxd_dhpcv6_client.c* files can be copied into this directory.

Using the NetX Duo DHCPv6 Client

The application code must include *nxd_dhcpv6_client.h* after it includes *tx_api.h* and *nx_api.h*, to use DHCPv6 Client, ThreadX and NetX Duo services, respectively. *nxd_dhcpv6_client.c* must be compiled in the project in the same manner as other application files and its object form must be linked along with the files of the application.

Client DHCP Unique Identifier (DUID)

The Client DUID uniquely defines each Client on a network. An application must create a Client DUID prior to requesting an IPv6 address from a Server. The Client DUID is automatically included in all messages to the Server. To create a DUID, the application calls the service *nx_dhcpv6_create_client_duid:*

```
UINT    nx_dhcpv6_create_client_duid(NX_DHCPV6 *dhcpv6_ptr,
                                     UINT duid_type,
                                     UINT hardware_type, ULONG time)
```

The application calls this service and specifies the type of DUID (link layer only, or link layer plus time. For link layer plus time DUIDs, this service will provide the time field if the time input is not specified.

For devices rebooting and wishing to use a previously assigned IPv6 address lease, the application must create the Client DUID as the one it used when assigned the IPv6 address. The link layer address is all that is needed to create a link layer Client DUID. This does not require previous non volatile memory storage if the device has access to the link layer address. For DUIDs of type time, the application must have access to the same time data used in the previous DUID creation and this does require non volatile memory. Clients that do not have any stable storage must not use DUIDs of type time

Client Identity Association for Non Temporary Addresses (IANA)

The application must create an IANA and optionally one or more IA addresses before requesting an IPv6 address. To do so, the application calls the *nx_dhcpv6_create_client_iana* service. To create an IA address option, the application calls the *nx_dhcpv6_add_client_ia* service with a requested IPv6 address and lifetime values as a hint to the Server.

The IANA and its IAs cumulatively define the Client IPv6 address assignment parameters:

Before starting the DHCPv6 Client, the DHCPv6 Client application creates an IANA using the *nx_dhcpv6_create_client_iana* service:

```
UINT    nx_dhcpv6_create_client_iana(NX_DHCPV6 *dhcpv6_ptr,
                                     UINT IA_ident, ULONG T1, ULONG T2)
```

It must also create one or more IAs using the *nx_dhcpv6_create_client_ia* service and requested IPv6 addresses before starting the DHCPv6 Client.

```
UINT    nx_dhcpv6_add_client_ia(NX_DHCPV6 *dhcpv6_ptr,
                                NXD_ADDRESS *ipv6_address,
                                ULONG preferred_lifetime,
                                ULONG valid_lifetime)
```

Note that the number of IA addresses the application creates cannot exceed the NX_DHCPV6_MAX_IA_ADDRESS parameter whose default value is 1.

The NetX Duo DHCPv6 Client supports *nx_dhcpv6_create_client_ia* for legacy DHCPv6 Client applications and which is identical to *nx_dhcpv6_add_client_ia* but developers are encouraged to use the *nx_dhcpv6_add_client_ia* service.

These services are demonstrated in the “Small Example System” elsewhere in this chapter.

Non Volatile Memory Considerations To Reuse IANAs and IAs

The application must save IANA parameters T1, T2, and the IANA identifier to non volatile memory if it wishes to use the same address(es) on rebooting. The application must also save its IA which includes its IPv6 address to non volatile memory.

The application must also store the time elapsed that it has been bound to its assigned IPv6 address lease(s) to non volatile memory if shutting down. It does this by calling the *nx_dhcpv6_get_time_accrued* service before it stops the DHCPv6 Client.

```
UINT nx_dhcpv6_get_time_accrued(NX_DHCPV6 *dhcpv6_ptr,
                                ULONG *time_accrued)
```

Assuming the application has an independent clock to track the time interval from when it stopped and restarted the DHCPv6 Client after a reboot, it adds to that elapsed time to the time accrued on the IPv6 lease before stopping. It now starts the Client thread task with the total elapsed time bound to the IPv6 lease as the `nv_time` input below:

```
UINT  nx_dhcpv6_start(NX_DHCPV6 *dhcpv6_ptr, ULONG nv_time)
```

From this point, the DHCPv6 Client thread task will take over monitoring the time accrued on the IPv6 lease for when to renew the lease.

Setting DHCPv6 Option Data

Before requesting an IPv6 lease, the application can request other network parameter data such as DNS server and time server. Some of these parameters have specific services. A few are shown below:

```
UINT  nx_dhcpv6_request_option_DNS_server(NX_DHCPV6 *dhcpv6_ptr,
                                          UINT enable)

UINT  nx_dhcpv6_request_option_time_server(NX_DHCPV6 *dhcpv6_ptr,
                                           UINT enable)
```

Initiating the IPv6 address Request

The application starts the DHCPv6 Client thread by calling the *nx_dhcpv6_start* service with a zero time input. To initiate the DHCPv6 protocol to request an IPv6 address, the application calls *nx_dhcpv6_request_solicit.*

If the application wishes to use a previously assigned IPv6 lease assigned, it calls *nx_dhcpv6_start* with a non zero time input. It should not call *nx_dhcpv6_request_solicit*.

Thereafter the application need do nothing more and the DHCPv6 Client will automatically monitor when it is time to renew or rebind an IPv6 address.

Small Example System

An example of how easy it is to use the NetX Duo DHCPv6 Client is described in the small example below using a DHCPv6 Client and a virtual “RAM” driver. This demo assumes a device with only a single physical network interface.

tx_application_define creates packet pool for the DHCPv6 Client to send DHCPv6 messages. It also creates an application thread and IP instance. It then enables UDP and ICMP on IP in lines 130-148. Then the DHCPv6 Client is created with state change (*dhcpv6_state_change_notify*) and server error (*dhcpv6_server_error_handler*) callback functions in line151.

In the Client thread entry function, *thread_client_entry*, the Client IP is set up with a link local address and enabled for IPv6 and ICMPv6 services on lines 202-217. Before starting the DHCPv6 Client, the application creates a Client DUID, an IANA option and an IA address option on lines219-303. The IA address option is optional if the Client wishes to request an IPv6 address and valid and preferred lifetimes from the Server. The Server may or may not grant the requested IPv6 address or lease times. The application may add more IA options (up to NX_DHCPV6_MAX_IA_ADDRESS) to be assigned multiple global addresses.

Lastly, the application sets various options to request network parameters in its messages to the DHCPv6 Server. The DHCPv6 Client task is started by calling *nx_dhcpv6_start* in line306, and the actual DHCPv6 protocol is started in the SOLICIT state with the call to *nx_dhcpv6_request_solicit* in line 317. The DHCPv6 Client then automatically handles the promotion of the Client state through the DHCPv6 protocol until it is bound to an address or an error occurs. During this time, the application waits for the protocol to complete, as well as the Duplicate Address Detection (DAD) to complete if the IP instance is configured for DAD (which is the default configuration).

After the tx_thread_sleep call, the application checks on the global parameters set in the state change callback to determine the success of both the DHCPv6 Client task to get assigned an IPv6 lease and if so, that the DAD check for uniqueness succeeded. This is done using the counters set up in the state change and server error callback functions. The application polls for non zero counts of `address_not_assigned, address_expired` and `server_errors` for `failed address assignment.` If the count of `bound_addresses` is non zero (at least one address successfully assigned), it checks for a non zero `address_failed_dad` for a failed DAD check. An explanation of the state change and server error callbacks follows:

The state change callback*, dhcpv6_state_change_notify*, the previous and current DHCPv6 Client state to determine if the Client received any valid Server responses:

- *dhcpv6_state_change_notify* checks for transitions directly from SOLICIT to INIT and if so it increments a counter for the DHCPv6 Client receiving no responses from the Server.

Next *dhcpv6_state_change_notify* checks if the Client was assigned (bound) to one or more IPv6 addresses:

- If the new state is BOUND, it increments a counter of addresses bound to the Client.

The *dhcpv6_state_change_notify* also checks for a failed DAD check:

- If the state transitions from DECLINE to INIT, the DHCPv6 Client has failed DAD check on one of its assigned addresses and increments the count of failed address assignments.

The last check by *dhcpv6_state_change_notify* in this example is for a successfully assigned address that passed the DAD check to fail to be renewed or rebind:

- If the state changes from REBIND to INIT, the Client did not get any responses to either its RENEW or REBIND requests and *dhcpv6_state_change_notify* increments its count of expired addresses.

The *dhcpv6_server_error_handler* if notified by the DHCPv6 Client task of an error status received from the Server increments the count of server errors.

Assuming all goes well, the application queries the DHCPv6 Client for address data including lease times. It gets a count of valid (successfully assigned) addresses by calling the *nx_dhcpv6_get_valid_ip_address_count* service and time to renew in the IANA (applies to all IA addresses assigned) by calling *nx_dhcpv6_get_iana_lease_time* on lines 372-392. It then queries the DHCPv6 Client for each of its IA options for IPv6 address and lease times by address index.

Some DHCPv6 Client services (*nx_dhcpv6_get_lease_time_data, nx_dhcpv6_get_IP_address*) do not require an address index as an input, and return DHCPv6 parameters for the primary Client global address. This is suitable for Clients with a single global IPv6 address when it calls *nx_dhcpv6_get_valid_ip_address_lease_time* in line 384.

The DHCPv6 Client configuration, NX_DHCPV6_CLIENT_RESTORE_STATE, t allows a system to restore a previously created DHCPv6 Client in Bound state between system reboots. Calling the *nx_dhcpv6_client_get_record* to get the DHCPv6 Client record between system reboots on line 434, calling the

nx_dhcpv6_client_restore_record to store the DHCPv6 Client record after system power up on line 525.

The application then releases the assigned addresses using the *nx_dhcpv6_request_release* service in line 552. To restart the application stops the DHCPv6 Client with the *nx_dhcpv6_client_stop* service in line 567 and clears all IPv6 addresses registered with the IP instance that were configured throughthe DHCPv6 Client. It does this by calling *nx_dhcpv6_reinitialize* in line 578. Then it restarts the DHCPv6 Client task with *nx_dhcpv6_start* and *nx_dhcpv6_request_solicit* services as before.

The DHCPv6 Client is deleted with the call to *nx_dhcpv6_delete* in line626. Note that it does not delete the packet pool it created for the DHCPv6 Client because this packet pool is also used by the IP instance. Otherwise it should delete the packet pool if it has no further use for it using the NetX Duo *nx_packet_pool_delete* service.

```
  1 /* This is a small demo of the NetX Duo DHCPv6 Client for the high-performance NetX Duo stack.
*/
  2
  3 #include    <stdio.h>
  4 #include   "tx_api.h"
  5 #include   "nx_api.h"
  6 #include   "nxd_dhcpv6_client.h"
  7
  8 #ifdef FEATURE_NX_IPV6
  9 #define     DEMO_STACK_SIZE         2048
 10
 11 /* Set the client address, and request these address from DHCPv6 Server.  */
 12 /*
 13 #define     NX_DHCPV6_REQUEST_IA_ADDRESS
 14 */
 15
 16 /* Set the list of DHCPv6 option data (timezone, DNS server, timer server, domain name)to get
from the DHCPv6 server.  */
 17
 18 #define     NX_DHCPV6_REQUEST_OPTION
 19
 20
 21 /* Add the fully qualified domain name to request whether the DHCPv6 server SHOULD or SHOULD
NOT perform the AAAA RR or DNS updates.  */
 22
 23 #define     NX_DHCPV6_REQUEST_FQDN_OPTION
 24
 25
 26 /* Define the ThreadX and NetX object control blocks...  */
 27
 28 NX_PACKET_POOL          pool_0;
 29 TX_THREAD               thread_client;
 30 NX_IP                   client_ip;
 31
 32 /* Define the Client and Server instances. */
 33
 34 NX_DHCPV6               dhcp_client;
 35
 36 /* Define the error counter used in the demo application...  */
 37 ULONG                   error_counter;
 38 CHAR                    *pointer;
 39
 40 /* Define thread prototypes.  */
```

```
 41 void    thread_client_entry(ULONG thread_input);
 42
 43 /***** Substitute your ethernet driver entry function here *********/
 44 extern VOID    _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
 45
 46 /* Declare DHCPv6 Client callbacks */
 47 VOID dhcpv6_state_change_notify(NX_DHCPV6 *dhcpv6_ptr, UINT old_state, UINT new_state);
 48 VOID dhcpv6_server_error_handler(NX_DHCPV6 *dhcpv6_ptr, UINT op_code, UINT status_code, UINT
message_type);
 49
 50 /* Set up globals for tracking changes to DHCPv6 Client from callback services. */
 51 UINT state_changes = 0;
 52 UINT address_expired = 0;
 53 UINT address_failed_dad = 0;
 54 UINT bound_addresses = 0;
 55 UINT address_not_assigned = 0;
 56 UINT server_errors = 0;
 57
 58 /* Define some DHCPv6 parameters. */
 59
 60 #define DHCPV6_IANA_ID      0xC0DEDBAD
 61 #define DHCPV6_T1           NX_DHCPV6_INFINITE_LEASE
 62 #define DHCPV6_T2           NX_DHCPV6_INFINITE_LEASE
 63 #define DHCPV6_RENEW_TIME   NX_DHCPV6_INFINITE_LEASE
 64 #define DHCPV6_REBIND_TIME  NX_DHCPV6_INFINITE_LEASE
 65 #define PACKET_PAYLOAD      500
 66 #define PACKET_POOL_SIZE    (5*PACKET_PAYLOAD)
 67
 68 /* Define main entry point.  */
 69
 70 int main()
71 {
 72
 73     /* Enter the ThreadX kernel.  */
 74     tx_kernel_enter();
 75 }
 76
 77
 78 /* Define what the initial system looks like.  */
 79
 80 void    tx_application_define(void *first_unused_memory)
 81 {
 82
 83 UINT    status;
 84
 85     /* Setup the working pointer.  */
 86     pointer =  (CHAR *) first_unused_memory;
 87
 88     /* Create the Client thread.  */
 89     status = tx_thread_create(&thread_client, "Client thread", thread_client_entry, 0,
 90                               pointer, DEMO_STACK_SIZE, 8, 8, TX_NO_TIME_SLICE, TX_AUTO_START);
 91
 92     /* Check for IP create errors.  */
 93     if (status)
 94     {
 95         error_counter++;
 96         return;
 97     }
 98
 99     pointer =  pointer + DEMO_STACK_SIZE;
100
101     /* Initialize the NetX system.  */
102     nx_system_initialize();
103
104     /* Create a packet pool.  */
105     status =  nx_packet_pool_create(&pool_0, "NetX Main Packet Pool", 1024,  pointer,
PACKET_POOL_SIZE);
106
107     pointer = pointer + PACKET_POOL_SIZE;
```

```
108
109     /* Check for pool creation error.  */
110     if (status)
111     {
112         error_counter++;
113         return;
114     }
115
116     /* Create a Client IP instance.  */
117     status = nx_ip_create(&client_ip, "Client IP", IP_ADDRESS(0, 0, 0, 0),
118                           0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
119                           pointer, 2048, 1);
120
121     pointer =  pointer + 2048;
122
123     /* Check for IP create errors.  */
124     if (status)
125     {
126         error_counter++;
127         return;
128     }
129
130     /* Enable UDP traffic for sending DHCPv6 messages.  */
131     status =  nx_udp_enable(&client_ip);
132
133     /* Check for UDP enable errors.  */
134     if (status)
135     {
136         error_counter++;
137         return;
138     }
139
140     /* Enable ICMP.  */
141     status =  nx_icmp_enable(&client_ip);
142
143     /* Check for ICMP enable errors.  */
144     if (status)
145     {
146         error_counter++;
147         return;
148     }
149
150     /* Create the DHCPv6 Client. */
151     status =  nx_dhcpv6_client_create(&dhcp_client, &client_ip, "DHCPv6 Client",
152                                       &pool_0, pointer, 2048, dhcpv6_state_change_notify,
153                                       dhcpv6_server_error_handler);
154
155     /* Check for errors.  */
156     if (status)
157     {
158         error_counter++;
159         return;
160     }
161
162     /* Update the stack pointer because we need it again. */
163     pointer = pointer + 2048;
164
165     /* Yield control to DHCPv6 threads and ThreadX. */
166     return;
167 }
168
169
170 /* Define the Client host application thread. */
171
172 void    thread_client_entry(ULONG thread_input)
173 {
174
175 UINT        status;
176 ULONG       T1, T2;
```

```
177 UINT        address_count;
178 UINT        address_index = 0;
179 NXD_ADDRESS valid_ipv6_address;
180 ULONG       preferred_lifetime;
181 ULONG       valid_lifetime;
182 UINT        ia_count = 1;
183
184 #ifdef NX_DHCPV6_REQUEST_IA_ADDRESS
185 NXD_ADDRESS ipv6_address;
186 #endif
187
188 #ifdef NX_DHCPV6_REQUEST_OPTION
189 UCHAR       buffer[200];
190 NXD_ADDRESS dns_server;
191 #endif
192
193 #ifdef NX_DHCPV6_CLIENT_RESTORE_STATE
194 ULONG       current_time;
195 ULONG       elapsed_time;
196 NX_DHCPV6_CLIENT_RECORD dhcpv6_client_record;
197 #endif
198
199
200     state_changes = 0;
201
202     /* Establish the link local address for the host. The RAM driver creates
203        a virtual MAC address of 0x1122334456. */
204     status = nxd_ipv6_address_set(&client_ip, 0, NX_NULL, 10, NULL);
205
206     if (status)
207     {
208         error_counter++;
209         return;
210     }
211
212     /* Let NetX Duo get initialized. */
213     tx_thread_sleep(50);
214
215     /* Enable the Client IP for IPv6 and ICMPv6 services. */
216     nxd_ipv6_enable(&client_ip);
217     nxd_icmp_enable(&client_ip);
218
219     /* Create a Link Layer Plus Time DUID for the DHCPv6 Client. Set time ID field
220        to NULL; the DHCPv6 Client API will supply one. */
221     status = nx_dhcpv6_create_client_duid(&dhcp_client, NX_DHCPV6_DUID_TYPE_LINK_TIME,
222                                           NX_DHCPV6_HW_TYPE_IEEE_802, 0);
223
224     if (status != NX_SUCCESS)
225     {
226         error_counter++;
227         return;
228     }
229
230     /* Create the DHCPv6 client's Identity Association (IA-NA) now.
231
232        Note that if this host had already been assigned in IPv6 lease, it
233        would have to use the assigned T1 and T2 values in loading the DHCPv6
234        client with an IANA block.
235     */
236     status = nx_dhcpv6_create_client_iana(&dhcp_client, DHCPV6_IANA_ID, DHCPV6_T1,  DHCPV6_T2);
237
238     if (status != NX_SUCCESS)
239     {
240         error_counter++;
241         return;
242     }
243
244 #ifdef NX_DHCPV6_REQUEST_IA_ADDRESS
245     memset(&ipv6_address,0x0, sizeof(NXD_ADDRESS));
```

```
246     ipv6_address.nxd_ip_version = NX_IP_VERSION_V6;
247     ipv6_address.nxd_ip_address.v6[0] = 0x3ffe0501;
248     ipv6_address.nxd_ip_address.v6[1] = 0xffff0100;
249     ipv6_address.nxd_ip_address.v6[2] = 0x00000000;
250     ipv6_address.nxd_ip_address.v6[3] = 0x0000abcd;
251
252     /* Create an IA address option.
253         Note that if this host had already been assigned in IPv6 lease, it
254         would have to use the assigned IPv6 address, preferred and valid lifetime
255         values in loading the DHCPv6 Client with an IA block.
256     */
257     status = nx_dhcpv6_add_client_ia(&dhcp_client, &ipv6_address,DHCPV6_RENEW_TIME,
DHCPV6_REBIND_TIME);
258
259     if (status != NX_SUCCESS)
260     {
261         error_counter++;
262         return;
263     }
264
265     /* If the DHCPv6 Client is configured for a maximum number of IA addresses
266        greater than 1, we can add another IA address if the device requires
267        multiple global IPv6 addresses.  */
268     if(NX_DHCPV6_MAX_IA_ADDRESS >= 2)
269     {
270         memset(&ipv6_address,0x0, sizeof(NXD_ADDRESS));
271         ipv6_address.nxd_ip_version = NX_IP_VERSION_V6;
272         ipv6_address.nxd_ip_address.v6[0] = 0x3ffe0501;
273         ipv6_address.nxd_ip_address.v6[1] = 0xffff0100;
274         ipv6_address.nxd_ip_address.v6[2] = 0x00000000;
275         ipv6_address.nxd_ip_address.v6[3] = 0x00001234;
276
277         /* Add another  IA address option.  */
278         status = nx_dhcpv6_add_client_ia(&dhcp_client, &ipv6_address, DHCPV6_RENEW_TIME,
DHCPV6_REBIND_TIME);
279
280         if (status != NX_SUCCESS)
281         {
282             error_counter++;
283             return;
284         }
285     }
286 #endif /* NX_DHCPV6_REQUEST_IA_ADDRESS  */
287
288 #ifdef NX_DHCPV6_REQUEST_OPTION
289     /* Set the list of DHCPv6 option data to get from the DHCPv6 server if needed. */
290     nx_dhcpv6_request_option_timezone(&dhcp_client, NX_TRUE);
291     nx_dhcpv6_request_option_DNS_server(&dhcp_client, NX_TRUE);
292     nx_dhcpv6_request_option_time_server(&dhcp_client, NX_TRUE);
293     nx_dhcpv6_request_option_domain_name(&dhcp_client, NX_TRUE);
294 #endif /* NX_DHCPV6_REQUEST_OPTION */
295
296
297 #ifdef NX_DHCPV6_REQUEST_FQDN_OPTION
298     /* Set the DHCPv6 Client FQDN option.
299        operation: NX_DHCPV6_CLIENT_DESIRES_UPDATE_AAAA_RR         DHCPv6 Client choose to
updating the FQDN-to-IPv6 address mapping for FQDN and address(es) used by the client.
300                   NX_DHCPV6_CLIENT_DESIRES_SERVER_DO_DNS_UPDATE   DHCPv6 Client choose to
updating the FQDN-to-IPv6 address mapping for FQDN and address(es) used by the client to the
server.
301                   NX_DHCPV6_CLIENT_DESIRES_NO_SERVER_DNS_UPDATE   DHCPv6 Client choose to
request that the server perform no DNS updatest on its behalf.  */
302     nx_dhcpv6_request_option_FQDN(&dhcp_client, "DHCPv6-Client",
NX_DHCPV6_CLIENT_DESIRES_UPDATE_AAAA_RR);
303 #endif /* NX_DHCPV6_REQUEST_FQDN_OPTION */
304
305     /* Start up the NetX DHCPv6 Client thread task. */
306     status =  nx_dhcpv6_start(&dhcp_client);
307
```

```
308     /* Check for errors. */
309     if (status != NX_SUCCESS)
310     {
311
312         error_counter++;
313         return;
314     }
315
316     /* Start the DHCPv6 by sending a Solicit message out on the network. */
317     status = nx_dhcpv6_request_solicit(&dhcp_client);
318
319     /* Check status.  */
320     if (status != NX_SUCCESS)
321     {
322
323         error_counter++;
324         return;
325     }
326
327     /* Is the DHCPv6 Client request for address assignment successfully started? */
328     if (status == NX_SUCCESS)
329     {
330
331         /* If Duplicate Address Detection (DAD) is enabled in NetX Duo, e.g.
#NXDUO_DISABLE_DAD
332            not defined, allow time for NetX Duo to verify the address is unique on our network.
333          */
334         tx_thread_sleep(500);
335
336         /* Check the bound address.  */
337         if (bound_addresses != ia_count)
338         {
339
340             /* Attempt to find out why DHCPv6 failed, where...*/
341
342             if (server_errors > 0)
343             {
344                 /* Actually you would compare server_error count with number of IA's added
345                    to determine if any addresses were assigned. */
346                 printf("Server error, not all address assigned\n");
347             }
348
349             if (address_not_assigned > 0)
350             {
351                 /* Actually you would compare address not assigned count with number of IA's
added
352                    to determine if any addresses were assigned. */
353
354                 printf("No servers responded to some or all of our IAs\n");
355             }
356
357         }
358
359         /* Regardless if the DHCPv6 Client achieved a bound state, check for DAD
360            failures. */
361         if (address_failed_dad > 0)
362         {
363             /* Actually you would compare failed dad count with number of IA's added
364                to determine if any addresses were assigned. */
365
366             printf("Some or all of our IAs failed DAD\n");
367
368         }
369
370         /* Successfully assigned IPv6 addresses! */
371
372         /* Get the count of valid IPv6 address obtained by DHCPv6.  */
373         status = nx_dhcpv6_get_valid_ip_address_count(&dhcp_client, &address_count);
374
```

```
375         /* Check status.  */
376         if (status != NX_SUCCESS)
377         {
378             error_counter++;
379         }
380
381         /* Get the IPv6 address and related lifetimes by address index. This index is the
382            index into the DHCPv6 Client address table.  Not to be confused with the IP
383            instance address table! */
384         status = nx_dhcpv6_get_valid_ip_address_lease_time(&dhcp_client, address_index,
385                                                            &valid_ipv6_address,
&preferred_lifetime,
386                                                            &valid_lifetime);
387
388         /* Check status.  */
389         if (status != NX_SUCCESS)
390         {
391             error_counter++;
392         }
393
394         /* Get the IANA options for when to start renew/rebind requests. These time
395            parameters are the same for all IPv6 addresses assigned in the Client
396            e.g. IANA returned from Server.  */
397         status = nx_dhcpv6_get_iana_lease_time(&dhcp_client, &T1, &T2);
398
399         /* Check status.  */
400         if (status != NX_SUCCESS)
401         {
402             error_counter++;
403         }
404
405         /*************************************************************************/
406         /* These are 'legacy' DHCPv6 services and are for the most part identical to the
services
407            above except they default to the primary global IPv6 address regardless if the
408            Client was assigned more than one global IPv6 address. */
409
410         /* Now check the assigned lease times. */
411         status = nx_dhcpv6_get_lease_time_data(&dhcp_client, &T1, &T2,
412                                                &preferred_lifetime, &valid_lifetime);
413
414         /* Check status.  */
415         if (status != NX_SUCCESS)
416         {
417             error_counter++;
418         }
419
420         /* Get the IP address.  */
421         status = nx_dhcpv6_get_IP_address(&dhcp_client, &valid_ipv6_address);
422
423         /* Check status.  */
424         if (status != NX_SUCCESS)
425         {
426             error_counter++;
427         }
428
429         /* Bound state.  */
430
431 #ifdef NX_DHCPV6_CLIENT_RESTORE_STATE
432
433         /* Get the DHCPv6 Client record.  */
434         nx_dhcpv6_client_get_record(&dhcp_client, &dhcpv6_client_record);
435
436         /* Delete DHCPv6 instance.  */
437         nx_dhcpv6_client_delete(&dhcp_client);
438
439         /* Delete IP instance.  */
440         status = nx_ip_delete(&client_ip);
441
```

```
442         /* Check for error.  */
443         if (status)
444         {
445             error_counter++;
446             return;
447         }
448
449         /* Create a Client IP instance.  */
450         status = nx_ip_create(&client_ip, "Client IP", IP_ADDRESS(0, 0, 0, 0),
451                               0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
452                               pointer, 2048, 1);
453
454         pointer =  pointer + 2048;
455
456         /* Check for IP create errors.  */
457         if (status)
458         {
459             error_counter++;
460             return;
461         }
462
463         /* Enable UDP traffic for sending DHCPv6 messages.  */
464         status =  nx_udp_enable(&client_ip);
465
466         /* Check for UDP enable errors.  */
467         if (status)
468         {
469             error_counter++;
470             return;
471         }
472
473         /* Enable ICMP.  */
474         status =  nx_icmp_enable(&client_ip);
475
476         /* Check for ICMP enable errors.  */
477         if (status)
478         {
479             error_counter++;
480             return;
481         }
482
483         /* Enable the Client IP for IPv6 and ICMPv6 services. */
484         status = nxd_ipv6_enable(&client_ip);
485         status += nxd_icmp_enable(&client_ip);
486
487         /* Check for IPv6 and ICMPv6 enable errors.  */
488         if (status)
489         {
490             error_counter++;
491             return;
492         }
493
494         /* Establish the link local address for the host. The RAM driver creates
495            a virtual MAC address of 0x1122334456. */
496         status = nxd_ipv6_address_set(&client_ip, 0, NX_NULL, 10, NULL);
497
498         if (status)
499         {
500             error_counter++;
501             return;
502         }
503
504         /* If Duplicate Address Detection (DAD) is enabled in NetX Duo, e.g.
#NXDUO_DISABLE_DAD
505            not defined, allow time for NetX Duo to verify the address is unique on our network.
506          */
507         tx_thread_sleep(500);
508
509         /* Create the DHCPv6 Client. */
```

```
510         status =  nx_dhcpv6_client_create(&dhcp_client, &client_ip, "DHCPv6 Client",
511                                           &pool_0, pointer, 2048, dhcpv6_state_change_notify,
512                                           dhcpv6_server_error_handler);
513
514         /* Check for errors.  */
515         if (status)
516         {
517             error_counter++;
518             return;
519         }
520
521         /* Update the stack pointer because we need it again. */
522         pointer = pointer + 2048;
523
524         /* Restore the DHCPv6 record.  */
525         nx_dhcpv6_client_restore_record(&dhcp_client, &dhcpv6_client_record, 5);
526
527         /* Resume the DHCPv6 service.  */
528         nx_dhcpv6_resume(&dhcp_client);
529 #endif
530
531
532 #ifdef NX_DHCPV6_REQUEST_OPTION
533
534         /* Get the DNS Server address.  */
535         nx_dhcpv6_get_DNS_server_address(&dhcp_client, 0, &dns_server);
536
537         /* Get the domain name.  */
538         memset(buffer, 0, sizeof(buffer));
539
540         nx_dhcpv6_get_other_option_data(&dhcp_client, NX_DHCPV6_DOMAIN_NAME_OPTION, buffer,
200); // Try to get DNS info got from DHCPv6 Server
541
542         /* Get the domain name.  */
543         memset(buffer, 0, sizeof(buffer));
544
545         /* Get the time zone.  */
546         nx_dhcpv6_get_other_option_data(&dhcp_client, NX_DHCPV6_NEW_POSIX_TIMEZONE_OPTION,
buffer, 200); // Try to get DNS info got from DHCPv6 Server
547 #endif
548
549         /* At some point, we may wish to release the IPv6 address lease e.g. the device
550            is leaving the network or powering down. In that case we inform the
551            DHCPv6 Server that we are releasing the address lease. */
552         status = nx_dhcpv6_request_release(&dhcp_client);
553
554         /* Check status.  */
555         if (status != NX_SUCCESS)
556         {
557
558             error_counter++;
559             return;
560         }
561
562         /* Send the release message.  */
563         tx_thread_sleep(100);
564     }
565
566     /* Stopping the Client task. */
567     status = nx_dhcpv6_stop(&dhcp_client);
568
569     /* Check status.  */
570     if (status != NX_SUCCESS)
571     {
572
573         error_counter++;
574         return;
575     }
576
```

```
577     /* Clear the previously assigned IPv6 addresses from the Client and IP address table.  */
578     status = nx_dhcpv6_reinitialize(&dhcp_client);
579
580     /* Check status.  */
581     if (status != NX_SUCCESS)
582     {
583
584         error_counter++;
585         return;
586     }
587
588     /* Start up the Client task again. */
589     status = nx_dhcpv6_start(&dhcp_client);
590
591     /* Check status.  */
592     if (status != NX_SUCCESS)
593     {
594
595         error_counter++;
596         return;
597     }
598
599     /* Begin the request process by sending a Solicit message with the IA created above
600        with our preferred IPv6 address. */
601     status = nx_dhcpv6_request_solicit(&dhcp_client);
602     /* Check status.  */
603     if (status != NX_SUCCESS)
604     {
605
606         error_counter++;
607         return;
608     }
609
610     /* Wait a bit before releasing the IP address and terminating the client. */
611     tx_thread_sleep(500);
612
613     /* Ok, lets stop the application. Again we DO NOT plan
614        to keep the IPv6 address we were assigned and need to release it
615        back to the DHCPv6 server. */
616     status = nx_dhcpv6_request_release(&dhcp_client);
617
618     /* Check for error. */
619     if (status != NX_SUCCESS)
620     {
621         error_counter++;
622     }
623
624     /* Now delete the DHCPv6 client and release ThreadX and
625        NetX Duo resources back to the system. */
626     nx_dhcpv6_client_delete(&dhcp_client);
627
628
629     return;
630
631 }
632
633
634 /* This is the notification from the DHCPv6 Client task that it has changed
635    state in the DHCPv6 protocol, for example getting assigned an IPv6 lease and
636    achieving the bound state or an IPv6 lease expires and being reset to
637    the init state.
638 */
639 VOID dhcpv6_state_change_notify(NX_DHCPV6 *dhcpv6_ptr, UINT old_state, UINT new_state)
640 {
641
642
643     /* Increment state change counter. */
644     state_changes++;
645
```

```
646     /* Check if the Client attempted to request an IPv6 lease but no servers
647        responded. */
648     if ((old_state == NX_DHCPV6_STATE_SENDING_SOLICIT) && (new_state == NX_DHCPV6_STATE_INIT))
649     {
650
651         /* Indication that either DAD failed or IP lease expired. */
652         address_not_assigned++;
653     }
654
655     /* Check if the Client has been assigned an IPv6 lease. */
656     if (new_state == NX_DHCPV6_STATE_BOUND_TO_ADDRESS)
657     {
658         bound_addresses++;
659     }
660
661    /* Check if the Client was bound, but failed the uniqueness check
662        (Duplicate Address Detection) and was reset to the INIT state. */
663     if ((old_state == NX_DHCPV6_STATE_SENDING_DECLINE) && (new_state == NX_DHCPV6_STATE_INIT))
664     {
665
666         /* Indication that DAD failed on Client IA. */
667         address_failed_dad++;
668     }
669
670     /* Check if the Client was bound, attempted renew the lease but the
671        IPv6 address renewal/rebinding failed. */
672     if ((old_state == NX_DHCPV6_STATE_SENDING_REBIND) && (new_state == NX_DHCPV6_STATE_INIT))
673     {
674
675         /* Indication that the IP lease expired. */
676         address_expired++;
677     }
678
679
680
681     /* Other checks are possible. */
682
683 }
684
685 /* This is the notification from the DHCPv6 Client task that it received an error
686    from the server (status code) in response to the Client's last DHCPv6 message.
687 */
688
689 VOID dhcpv6_server_error_handler(NX_DHCPV6 *dhcpv6_ptr, UINT op_code, UINT status_code, UINT
message_type)
690 {
691
692     /* Increment the server error count. */
693     server_errors++;
694
695     /* This should distinguish between receiving a server error and no server
696        available to assign the Client an IPv6 address if the Client fails
697        to get assigned an address. */
698 }
699
700 #endif /* FEATURE_NX_IPV6 */
```

Chapter 3
NetX Duo DHCPv6 Configuration Options

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

Define	Meaning
NX_DHCPV6_THREAD_PRIORITY	Priority of the Client thread. By default, this value specifies that the Client thread runs at priority 2.
NX_DHCPV6_MUTEX_WAIT	Time out option for obtaining an exclusive lock on a DHCPv6 Client mutex The default value is TX_WAIT_FOREVER.
NX_DHCPV6_TICKS_PER_SECOND	Ratio of ticks to seconds. This is processor dependent. The default value is 100.
NX_DHCPV6_IP_LIFETIME_TIMER_INTERVAL	Time interval in seconds at which the IP lifetime timer updates the length of time the current IP address has been assigned to the Client. By default, this value is 1.
NX_DHCPV6_SESSION_TIMER_INTERVAL	Time interval in seconds at which the session timer updates the length of time the Client has been in session communicating with the Server. By default, this value is 1.
NX_DHCPV6_MAX_IA_ADDRESS	The maximum number of IA addresses that can be added to the Client record. The default value is 1.

NX_DHCPV6_NUM_DNS_SERVERS Number of DNS servers to store to the client record. The default value is 2.

NX_DHCPV6_NUM_TIME_SERVERS Number of time servers to store to the client record. The default value is 1.

NX_DHCPV6_DOMAIN_NAME_BUFFER_SIZE
Size of the buffer in the Client record to hold the client's network domain name. The default value is 30.

NX_DHCPV6_TIME_ZONE_BUFFER_SIZE
Size of the buffer in the Client record to hold the Client's time zone. The default value is 10.

NX_DHCPV6_MAX_MESSAGE_SIZE Size of the buffer in the Client record to hold the option status message in a Server reply. The default value is 100 bytes.

NX_DHCPV6_PACKET_TIME_OUT Time out in seconds for allocating a packet from the Client packet pool. The default value is 3 seconds.

NX_DHCPV6_TYPE_OF_SERVICE This defines the type of service for UDP packet transmission from the DHCPv6 Client socket. The default value is **NX_IP_NORMAL**.

NX_DHCPV6_TIME_TO_LIVE The number of times a Client packet is forwarded by a network router before the packet is discarded. The default value is 0x80.

NX_DHCPV6_QUEUE_DEPTH Specifies the number of packets to keep in the Client UDP socket receive queue

before NetX Duo discards packets. The default value is 5.

DHCPv6 Message Transmission

There are a set of DHCPv6 Client options for setting parameters on DHCPv6 message transmission. These are:

- initial timeout
- maximum delay on the first transmission
- maximum retransmission timeout
- maximum number of retransmissions
- maximum duration to wait for server response

These parameters apply to each of the DHCPv6 Client messages:

SOLICIT
REQUEST
RENEW
REBIND
RELEASE
DECLINE
CONFIRM
INFORM

The following is a complete list of these configurable options and their default values:

```
NX_DHCPV6_FIRST_SOL_MAX_DELAY                    (1 * NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_INIT_SOL_TRANSMISSION_TIMEOUT          (1 * NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_SOL_RETRANSMISSION_TIMEOUT         (120 *
                                                  NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_SOL_RETRANSMISSION_COUNT           0
NX_DHCPV6_MAX_SOL_RETRANSMISSION_DURATION        0

NX_DHCPV6_INIT_REQ_TRANSMISSION_TIMEOUT          (1 * NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_REQ_RETRANSMISSION_TIMEOUT         (30 * NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_REQ_RETRANSMISSION_COUNT           10
NX_DHCPV6_MAX_REQ_RETRANSMISSION_DURATION        0

NX_DHCPV6_INIT_RENEW_TRANSMISSION_TIMEOUT        (10*NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_RENEW_RETRANSMISSION_TIMEOUT (600*
                                                  NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_RENEW_RETRANSMISSION_COUNT         0

NX_DHCPV6_INIT_REBIND_TRANSMISSION_TIMEOUT       (10*NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_REBIND_RETRANSMISSION_TIMEOUT (600*
                                                  NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_REBIND_RETRANSMISSION_COUNT        0

NX_DHCPV6_INIT_RELEASE_TRANSMISSION_TIMEOUT      (1*NX_DHCPV6_TICKS_PER_SECOND)
```

```
NX_DHCPV6_MAX_RELEASE_RETRANSMISSION_TIMEOUT    0
NX_DHCPV6_MAX_RELEASE_RETRANSMISSION_COUNT      5
NX_DHCPV6_MAX_RELEASE_RETRANSMISSION_DURATION 0

NX_DHCPV6_INIT_DECLINE_TRANSMISSION_TIMEOUT       (1*NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_DECLINE_RETRANSMISSION_TIMEOUT    0
NX_DHCPV6_MAX_DECLINE_RETRANSMISSION_COUNT      5
NX_DHCPV6_MAX_DECLINE_RETRANSMISSION_DURATION 0
NX_DHCPV6_FIRST_CONFIRM_MAX_DELAY                  (1*NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_INIT_CONFIRM_TRANSMISSION_TIMEOUT       (1*NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_CONFIRM_RETRANSMISSION_TIMEOUT   (4*NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_CONFIRM_RETRANSMISSION_COUNT      0
NX_DHCPV6_MAX_CONFIRM_RETRANSMISSION_DURATION 10

NX_DHCPV6_FIRST_INFORM_MAX_DELAY                  (1*NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_INIT_INFORM_TRANSMISSION_TIMEOUT       (1*NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_INFORM_RETRANSMISSION_TIMEOUT   (120*
                                                  NX_DHCPV6_TICKS_PER_SECOND)
NX_DHCPV6_MAX_INFORM_RETRANSMISSION_COUNT       0
NX_DHCPV6_MAX_INFORM_RETRANSMISSION_DURATION  0
```

For no limit on a retransmission timeout, set the message retransmission count to 0. For no limit on the number of times a DHCPv6 Client message is retransmitted (retries), set the message retransmission count to 0.

Note that regardless of length of timeout or number of retries, when an IPv6 address valid lifetime expires, it is removed from the IP address table and can no longer be used by the Client. The NetX Duo DHCPv6 Client will automatically begin sending SOLICIT messages requesting a new IPv6 address.

Chapter 4
NetX Duo DHCPv6 Client Services

This chapter contains a description of all NetX Duo DHCPv6Client services (listed below) in alphabetic order.

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

nx_dhcpv6_client_create
Create a DHCPv6 Client instance

nx_dhcpv6_client_delete
Delete a DHCPv6 Client instance

nx_dhcpv6_create_ client_duid
Create a DHCPv6 Client DUID

nx_dhcpv6 _add_client_ia
Add a DHCPv6 Client Identity Address (IA)

nx_dhcpv6 _create_client_ia
(*Legacy Add a DHCPv6 Client Identity Address (IA))*

nx_dhcpv6_create_client_iana
Create a DHCPv6 Client Identity Association for Non Temporary Addresses (IANA)

nx_dhcpv6_get_client_duid_time_id
Get the time ID from DHCPv6 Client DUID

nx_dhcpv6_client_set_interface
Set the Client network interface for communications with the DHCPv6 Server

nx_dhcpv6_get_IP_address
Get the global IPv6 address assigned to the DHCPv6 client

nx_dhcpv6_get_lease_time_data
Get T1, T2, valid and preferred lifetimes for the Client global IPv6 address

nx_dhcpv6_get_valid_ip_address_lease_time

Get T1, T2, valid and preferred lifetimes for the DHCPv6 Client IPv6 address by address index

nx_dhcpv6_get_iana_lease_time
Get T1 and T2 in the Identity Association (IANA) leased to the DHCPv6 Client

nx_dhcpv6_get_other_option_data
Get the specified option data e.g. domain name or time zone server

nx_dhcpv6_get_DNS_server_address
Get DNS Server address at the specified index into the DHCPv6 Client DNS server list

nx_dhcpv6_get_time_accrued
Get the time accrued the global IPv6 address lease has been bound to the DHCPv6 Client

nx_dhcpv6_get_time_server_address
Get Time Server address at the specified index into the DHCPv6 Client Time server list

nx_dhcpv6_get_valid_ip_address_count
Get the number of IPv6 addresses assigned to the DHCPv6 Client

nx_dhcpv6_reinitialize
Reinitialize the DHCPv6 for restarting the DHCPv6 Client state machine and rerunning the DHCPv6 protocol

nx_dhcpv6_request_confirm
Send a CONFIRM request to the Server

nx_dhcpv6_request_inform_request
Send an INFORM REQUEST message to the Server

nx_dhcpv6_request_release
Send a RELEASE request to the Server

nx_dhcpv6_request_option_DNS_server
Add the DNS server option to the Client option request data in request messages to the Server

nx_dhcpv6_request_option_FQDN

Add the FQDN option to the Client option request data in request messages to the Server

nx_dhcpv6_request_option_domain_name
Add the domain name option to the Client option request data in request messages to the Server

nx_dhcpv6_request_option_time_server
Add the time server option to the Client option request data in request messages to the Server

nx_dhcpv6_request_option_timezone
Add the time zone option to the Client option request data in request messages to the Server

nx_dhcpv6_request_solicit
Send a DHCPv6 SOLICIT request to any Server on the Client network (broadcast)

nx_dhcpv6_request_solicit_rapid
Send a DHCPv6 SOLICIT request to any Server on the Client network (broadcast) with the Rapid Commit option set

nx_dhcpv6_resume
Resume DHCPv6 Client processing

nx_dhcpv6_start
Start the DHCPv6 Client thread task. Note this is not equivalent to starting the DHCPv6 state machine and does not send a SOLICIT request

nx_dhcpv6_stop
Stop the DHCPv6 Client thread task

nx_dhcpv6_suspend
Suspend the DHCPv6 Client thread task

nx_dhcpv6_set_time_accrued
Set the time accrued on the global Client IPv6 address lease in the Client record.

nx_dhcpv6_client_create

Create a DHCPv6 client instance

Prototype

```
UINT  nx_dhcpv6_client_create(NX_DHCPV6 *dhcpv6_ptr,
                              NX_IP *ip_ptr, CHAR *name_ptr,
                              NX_PACKET_POOL *packet_pool_ptr,
                              VOID *stack_ptr, ULONG stack_size,
                              VOID (*dhcpv6_state_change_notify)
                                       (struct NX_DHCPV6_STRUCT *dhcpv6_ptr,
                                        UINT old_state, UINT new_state),
                              VOID (*dhcpv6_server_error_handler)
                                       (struct NX_DHCPV6_STRUCT *dhcpv6_ptr,
                                       UINT op_code, UINT status_code,
                                       UINT message_type))
```

Description

This service creates a DHCPv6 client instance including callback functions.

Input Parameters

dhcpv6_ptr Pointer to DHCPv6 control block
ip_ptr Pointer to Client IP instance
name_ptr Pointer to name for DHCPv6 instance
packet_pool_ptr Pointer to Client packet pool
stack_ptr Pointer to Client stack memory
stack_size Size of Client stack memory
dhcpv6_state_change_notify
Pointer to callback function invoked when the Client initiates a new DHCPv6 request to the server
dhcpv6_server_error_handler
Pointer to callback function invoked when the Client receives an error status from the server

Return Values

NX_SUCCESS	(0x00)	Successful Client create
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

```
/* Create a DHCPv6 client instance without specifying link local or preferred global IP
address.  */
status =  nx_dhcpv6_client_create(&dhcp_0, &ip_0, "DHCPv6 Client", &pool_0,
                                  NULL, NULL, pointer, 2048,
                                  dhcpv6_state_change_notify,
                                  dhcpv6_server_error_handler);

/* If status is NX_SUCCESS a DHCPv6 client instance was successfully
   created.  */
```

See Also

nx_dhcpv6_client_delete

nx_dhcpv6_client_delete

Delete a DHCPv6 Client instance

Prototype

```
UINT nx_dhcpv6_client_delete(NX_DHCPV6 *dhcpv6_ptr);
```

Description

This service deletes a previously created DHCPv6 client instance.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 client instance

Return Values

NX_SUCCESS	(0x00)	Successful DHCPv6 deletion
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

```
/* Delete a DHCPv6 client instance.  */
status =  nx_dhcpv6_client_delete(&my_dhcp);

/* If status is NX_SUCCESS the DHCPv6 client instance was successfully
   deleted.  */
```

See Also

nx_dhcpv6_client_create

nx_dhcpv6_client_set_interface

Sets Client's Network Interface for DHCPv6

Prototype

```
UINT    nx_dhcpv6_client_set_interface(NX_DHCPV6 *dhcpv6_ptr,
                                       UINT *interface_index)
```

Description

This service sets the Client's network interface for communicating with the DHCPv6 Server(s) to the specified input interface index.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
interface_index	Index indicating network interface

Return Values

NX_SUCCESS	(0x00)	Interface successfully set
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_INVALID_INTERFACE	(0x4C)	Invalid interface index input

Allowed From

Threads

Example

```
/* Set the client interface for DHCPv6 communication with the Server to the secondary
    interface (1). */

UINT index = 1;
status = nx_dhcpv6_client_set_interface(&dhcp_0, index);

/* If status is NX_SUCCESS, the Client successfully set the DHCPv6 network interface.  */
```

See Also

nx_dhcpv6_client _create, nx_dhcpv6_start

nx_dhcpv6_client_set_destination_address

Sets the destination address where DHCPv6 message should be sent to

Prototype

```
UINT    nx_dhcpv6_client_set_destination_address(NX_DHCPV6 *dhcpv6_ptr,
                                    NXD_ADDRESS *destination_address)
```

Description

This service sets the destination address where DHCPv6 message should be sent to. By default is ALL_DHCP_Relay_Agents_and_Servers(FF02::1:2).

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
destination_address	Destination address

Return Values

NX_SUCCESS	(0x00)	Interface successfully set
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Parament error

Allowed From

Threads

Example

```
/* Set the destination address where DHCPv6 message should be sent to. */
NXD_ADDRESS dest_address; /* Set the destination address.  */
status = nx_dhcpv6_client_set_destination_address(&dhcp_0, &dest_address);
/* If status is NX_SUCCESS, the Client successfully set the destination address.  */
```

See Also

nx_dhcpv6_client _create, nx_dhcpv6_start

nx_dhcpv6_create_client_duid

Create Client DUID object

Prototype

```
UINT    nx_dhcpv6_create_client_duid(NX_DHCPV6 *dhcpv6_ptr,
                                     UINT duid_type, UINT hardware_type,
                                     ULONG time)
```

Description

This service creates the Client DUID with the input parameters. If the time input is not supplied and the duid type indicates link layer with time, this function will supply a time which includes a randomizing factor for uniqueness. Vendor assigned (enterprise) duid types are not supported.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
duid_type	Type of DUID (hardware, enterprise etc)
hardware_type	Network hardware e.g. IEEE 802
time	Value used in creating unique identifier

Return Values

NX_SUCCESS	(0x00)	Successful Client DUID created
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input
NX_DHCPV6_UNSUPPORTED_DUID_TYPE	(0xE98)	DUID type unknown or not supported
NX_DHCPV6_UNSUPPORTED_DUID_HW_TYPE	(0xE99)	DUID hardware type unknown or not supported

Allowed From

Threads

Example

```
/* Create the Client DUID from the supplied input.  The time field is left NULL so the
    DHCPv6 client will provide one.  */
status = nx_dhcpv6_create_client_duid(&dhcp_0, NX_DHCPV6_DUID_TYPE_LINK_TIME,
                                      NX_DHCPV6_HW_TYPE_IEEE_802, 0)

/* If status is NX_SUCCESS the client DUID was successfully created.  */
```

See Also

nx_dhcpv6_create_client_ia, nx_dhcpv6_create_client_iana, nx_dhcpv6_create_server_duid

nx_dhcpv6_create_client_ia

Add an Identity Association to the Client

Prototype

```
UINT    nx_dhcpv6_create_client_ia(NX_DHCPV6 *dhcpv6_ptr,
                                   NXD_ADDRESS *ipv6_address,
                                   ULONG preferred_lifetime,
                                   ULONG valid_lifetime)
```

Description

This service is identical to the *nx_dhcpv6_add_client_ia* service. It adds a Client Identity Association by filling in the Client record with the supplied parameters. To request the maximum preferred and valid lifetimes, set these parameters to infinity. To add more than one IA to a DHCPv6 Client, set the NX_DHCPV6_MAX_IA_ADDRESS to a value higher than the default value of 1.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
ipv6_address	Pointer to NetX Duo IP address block
preferred_lifetime	Length of time before IP address is deprecated
valid_lifetime	Length of time before IP address is expired

Return Values

NX_SUCCESS	(0x00)	Successful Client IA added
NX_DHCPV6_IA_ADDRESS_ALREADY_EXIST	(0xEAF)	Duplicate IA address
NX_DHCPV6_REACHED_MAX_IA_ADDRESS	(0xEAE)	IA exceeds the max IAs Client can store
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_INVALID_IA_ADDRESS	(0xEA4)	Invalid (e.g. null) IA address in IA
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

```
/* Add an Client IA using the supplied input.   */
    status = nx_dhcpv6_create_client_ia(&dhcp_0, &ipv6_address,
                                        NX_DHCPV6_PREFERRED_LIFETIME,
                                        NX_DHCPV6_VALID_LIFETIME);

/* If status is NX_SUCCESS the client IA was successfully added.  */
```

See Also

nx_dhcpv6_add_client_duid, nx_dhcpv6_create_server_duid, nx_dhcpv6_create_client_iana

nx_dhcpv6_create_client_iana

Create an Identity Association (Non Temporary) for the Client

Prototype

```
UINT    nx_dhcpv6_create_client_iana(NX_DHCPV6 *dhcpv6_ptr,
                                     UINT IA_ident, ULONG T1, ULONG T2)
```

Description

This service creates a Client Non Temporary Identity Association (IANA) from the supplied parameters. To set the T1 and T2 times to maximum (infinity) in the DHCPv6 Client requests, set these parameters to NX_DHCPV6_INFINITE_LEASE. Note that a Client has only one IANA.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
IA_ident	Identity Association unique identifier
T1	When the Client must start the IPv6 address renewal
T2	When the Client must start theIPv6 address rebinding

Return Values

NX_SUCCESS	(0x00)	Successfully created the IANA
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

```
/* Create the Client IANA from the supplied input.  */
status = nx_dhcpv6_create_client_iana(&dhcp_0, DHCPV6_IA_ID, DHCPV6_T1,
                                      DHCPV6_T2);

/* If status is NX_SUCCESS the client IANA was successfully created.  */
```

See Also

nx_dhcpv6_create_client_duid, nx_dhcpv6_create_server_duid, nx_dhcpv6_add_client_ia

nx_dhcpv6_add_client_ia

Add an Identity Association to the Client

Prototype

```
UINT    nx_dhcpv6_add_client_ia(NX_DHCPV6 *dhcpv6_ptr,
                                NXD_ADDRESS *ipv6_address,
                                ULONG preferred_lifetime,
                                ULONG valid_lifetime)
```

Description

This service adds a Client Identity Association by filling in the Client record with the supplied parameters. To request the maximum preferred and valid lifetimes, set these parameters to infinity. To add more than one IA to a DHCPv6 Client, set the NX_DHCPV6_MAX_IA_ADDRESS to a value higher than the default value of 1.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
ipv6_address	Pointer to NetX Duo IP address block
preferred_lifetime	Length of time before IP address is deprecated
valid_lifetime	Length of time before IP address is expired

Return Values

NX_SUCCESS	(0x00)	Successful Client IA added
NX_DHCPV6_IA_ADDRESS_ALREADY_EXIST	(0xEAF)	Duplicate IA address
NX_DHCPV6_REACHED_MAX_IA_ADDRESS	(0xEAE)	IA exceeds the max IAs Client can store
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_INVALID_IA_ADDRESS	(0xEA4)	Invalid (e.g. null) IA address in IA
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

```
/* Add an Client IA using the supplied input.   */
status = nx_dhcpv6_add_client_ia(&dhcp_0, &ipv6_address, NX_DHCPV6_PREFERRED_LIFETIME,
                                 NX_DHCPV6_VALID_LIFETIME);

/* If status is NX_SUCCESS the client IA was successfully added.  */
```

See Also

nx_dhcpv6_create_client_duid, nx_dhcpv6_create_server_duid, nx_dhcpv6_create_client_iana

nx_dhcpv6_get_client_duid_time_id

Retrieves time ID from Client DUID

Prototype

```
UINT nx_dhcpv6_get_client_duid_time_id(NX_DHCPV6 *dhcpv6_ptr,
                                       ULONG *time_id)
```

Description

This service retrieves the time ID field from the Client DUID. If the application must first call *nx_dhcpv6_create_client_duid*, to fill in the Client DUID in the DHCPv6 Client instance or it will have a null value for this field. The intent is for the application to save this data and present the same Client DUID to the server, including the time field, across reboots.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
time_id	Pointer to Client DUID time field

Return Values

NX_SUCCESS	(0x00)	IP lease data successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Retrieve the time ID from the Client DUID.  */
status = nx_dhcpv6_get_client_duid_time_id(&dhcp_0, &time_ID);

/* If status is NX_SUCCESS the time ID was retrieved. */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_time_lease_data, nx_dhcpv6_get_other_option_data, nx_dhcpv6_get_time_accrued

nx_dhcpv6_get_IP_address

Retrieves Client's global IPv6 address

Prototype

```
UINT    nx_dhcpv6_get_IP_address(NX_DHCPV6 *dhcpv6_ptr,
                                 NXD_ADDRESS *ip_address)
```

Description

This service retrieves the Client's global IPv6 address. If the Client does not have a valid address, an error status is returned. If a Client has more than one global IPv6 address, the primary IPv6 address is returned.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
ip_address	Pointer to IPv6 address

Return Values

NX_SUCCESS	(0x00)	IPv6 address successfully assigned
NX_DHCPV6_IA_ADDRESS_NOT_VALID	(0xEAD)	IPv6 address is not valid
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
UINT address_status;
UINT address_index;

/* Retrieve the client's assigned IP address.  */
status = nx_dhcpv6_get_IP_address(&dhcp_0, &ipv6_address);

/* If status is NX_SUCCESS the client IP address was assigned.  Now register it with NetX
    Duo on the primary interface (index zero). The address index is returned in the
    address_index field*/
status = nxd_ipv6_address_set(&ip_0, 0, &ipv6_address, 64, &address_index);
```

See Also

nx_dhcpv6_get_lease_time_data,nx_dhcpv6_get_client_duid_time_id,nx_dhcpv6_get_other_option_data, nx_dhcpv6_get_time_accrued

nx_dhcpv6_get_lease_time_data

Retrieves Client's IA address lease time data

Prototype

```
UINT nx_dhcpv6_get_lease_time_data(NX_DHCPV6 *dhcpv6_ptr, ULONG *T1,
                                   ULONG *T2, ULONG *preferred_lifetime,
                                   ULONG *valid_lifetime)
```

Description

This service retrieves the Client's global IA address time data. If the Client IA address status is invalid, time data is set to zero and a successful completion status is returned. If a Client has more than one global IPv6 address, the primary IA address data is returned.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
T1	Pointer to IA address renew time
T2	Pointer to IA address rebind time
preferred_lifetime	Pointer to time when IA address is deprecated
valid_lifetime	Pointer to time when IA address is expired

Return Values

NX_SUCCESS	(0x00)	IA lease data successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Retrieve the client's assigned IA lease data.  */
status = nx_dhcpv6_get_lease_time_data(&dhcp_0, &T1, &T2, &preferred_lifetime,
                                       &valid_lifetime);

/* If status is NX_SUCCESS the client IA address lease data was retrieved.  */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_client_duid_time_id, nx_dhcpv6_get_other_option_data, nx_dhcpv6_get_time_accrued, nx_dhcpv6_get_iana_lease_time

nx_dhcpv6_get_iana lease_time

Retrieve the Client's IANA lease time data

Prototype

```
UINT nx_dhcpv6_get_iana_lease_time(NX_DHCPV6 *dhcpv6_ptr, ULONG *T1,
                                   ULONG *T2)
```

Description

This service retrieves the Client's global IA-NA lease time data (T1 and T2). If none of the Client IA-NA addresses have a valid address status, time data is set to zero and a successful completion status is returned. If a Client has more than one global IPv6 address, the primary IA address data is returned.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
T1	Pointer to time for starting lease renewal
T2	Pointer to time for starting lease rebinding

Return Values

NX_SUCCESS	(0x00)	IANA lease data successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Retrieve the client's assigned IANA lease data.  */
status = nx_dhcpv6_get_iana_lease_time(&dhcp_0, &T1, &T2);

/* If status is NX_SUCCESS the client IA address lease data was retrieved.  */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_client_duid_time_id, nx_dhcpv6_get_other_option_data, nx_dhcpv6_get_time_accrued, nx_dhcpv6_get_lease_time_data

nx_dhcpv6_get_valid_ip_address_count

Retrieve a count of Client's valid IA addresses

Prototype

```
UINT nx_dhcpv6_get_valid_ip_address_count(NX_DHCPV6 *dhcpv6_ptr,
                                          UINT *address_count)
```

Description

This service retrieves the count of the Client's valid IPv6 addresses. A valid IPv6 address is bound (assigned) to the Client and registered with the IP instance.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
address_count	Pointer to address count to return

Return Values

NX_SUCCESS	(0x00)	IANA lease data successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
UINT address_count;

/* Retrieve the count of valid IA-NA addresses.  */
status = nx_dhcpv6_get_valid_ip_address_count(&dhcp_0, &address_count);

/* If status is NX_SUCCESS the client IA address count was retrieved.  */
```

nx_dhcpv6_get_valid_ip_address_lease_time

Retrieve the Client IA lease data by address index

Prototype

```
UINT nx_dhcpv6_get_valid_ip_address_lease_time(NX_DHCPV6 *dhcpv6_ptr,
                                               UINT address_index,
                                               NXD_ADDRESS *ip_address,
                                               ULONG *preferred_lifetime,
                                               ULONG *valid_lifetime)
```

Description

This service retrieves the Client's IA address lease data by address index. .

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
address_index	Pointer to IA IPv6 address
preferred_lifetime	Pointer to time when IA address is deprecated
valid_lifetime	Pointer to time when IA address is expired

Return Values

NX_SUCCESS	(0x00)	IANA lease data successfully retrieved
NX_DHCPV6_IA_ADDRESS_NOT_VALID	(0xEAD)	No valid IA address available
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
UINT address_count;
NXD_ADDRESS ip_address;

/* Retrieve the count of valid IA-NA addresses.  */
status = nx_dhcpv6_get_valid_ip_address_lease_time(&dhcp_0, &ip_address,
                                                   &preferred_lifetime,
                                                   &valid_lifetime);

/* If status is NX_SUCCESS the client IA lease data was retrieved.  */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_iana_lease_time, nx_dhcpv6_get_lease_time_data

nx_dhcpv6_get_DNS_server_address

Retrieves DNS Server address

Prototype

```
UINT  nx_dhcpv6_get_DNS_server_address(NX_DHCPV6 *dhcpv6_ptr, UINT index,
                                       NXD_ADDRESS *server_address)
```

Description

This service retrieves the DNS server IPv6 address data at the specified index in the Client list. If the list does not contain a server address at the index, an error is returned. The index may not exceed the size of the DNS Server list is specified by the user configurable option NX_DHCPV6_NUM_DNS_SERVERS.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
index	Index into the DNS Server list
server_address	Pointer to Server address buffer

Return Values

NX_SUCCESS	(0x00)	Address successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Retrieve the DNS server at the specified index in the list. */

UINT index = 0;
NXD_ADDRESS server_address;


  status = nx_dhcpv6_get_DNS_server_address(&dhcp_0, index, &server_address);

/* If status == NX_SUCCESS, the DNS server IP address successfully retrieved. */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_lease_time_data, nx_dhcpv6_get_time_accrued

nx_dhcpv6_get_other_option_data

Retrieves DHCPv6 option data

Prototype

```
UINT  nx_dhcpv6_get_other_option_data(NX_DHCPV6 *dhcpv6_ptr,
                                      UINT option_code, UCHAR *buffer)
```

Description

This service retrieves DHCPv6 option data from a DHCPv6 message for the specified option code.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
option code	Option code for which data to retrieve
buffer	Pointer to buffer to copy data to

Return Values

NX_SUCCESS	(0x00)	Option data successfully retrieved
NX_DHCPV6_UNKNOWN_OPTION	(0xEAB)	Unknown/unsupported option code
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Retrieve the option data specified by the input option code. */
status = nx_dhcpv6_get_other_option_data(&dhcp_0, option_code, buffer);

/* If status is NX_SUCCESS the option data was retrieved.  */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_lease_time_data, nx_dhcpv6_get_time_accrued

nx_dhcpv6_get_time_accrued

Retrieves time accrued on Client's IP address lease

Prototype

```
UINT nx_dhcpv6_get_time_accrued(NX_DHCPV6 *dhcpv6_ptr, ULONG *time_accrued)
```

Description

This service retrieves the time accrued on the Client's IPv6 address lease. The function checks all the IPv6 addresses assigned to the Client for the first valid address. If no valid addresses are found, a zero value for time accrued is returned.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
time_accrued	Pointer to time accrued in IP lease

Return Values

NX_SUCCESS	(0x00)	Accrued time successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Retrieve time accrued time on the Client address lease.  */
status = nx_dhcpv6_get_time_accrued(&dhcp_0, &time_accrued);

/* If status is NX_SUCCESS the time accrued on the client IP address lease was retrieved.
   */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_other_option_data, nx_dhcpv6_get_lease_time_data, nx_dhcpv6_set_time_accrued

nx_dhcpv6_get_time_server_address

Retrieves Time Server address

Prototype

```
UINT  nx_dhcpv6_get_time_server_address(NX_DHCPV6 *dhcpv6_ptr, UINT index,
                                        NXD_ADDRESS *server_address)
```

Description

This service retrieves the Time server IPv6 address data at the specified index in the Client list. If the list does not contain a server address at the index, an error is returned. The index may not exceed the size of the Time Server list is specified by the user configurable option NX_DHCPV6_NUM_TIME_SERVERS.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
index	Index into the Time Server list
server_address	Pointer to Server address buffer

Return Values

NX_SUCCESS	(0x00)	Address successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Retrieve the Time server at the specified index in the list. */

UINT index = 0;
NXD_ADDRESS server_address;


  status = nx_dhcpv6_get_time_server_address(&dhcp_0, index, &server_address);

/* If status == NX_SUCCESS, the Time server IP address successfully retrieved. */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_lease_time_data, nx_dhcpv6_get_time_accrued, nx_dhcpv6_get_DNS_server_address

nx_dhcpv6_reinitialize

Remove the Client IP address from the IP table

Prototype

```
UINT nx_dhcpv6_reinitialize(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service reinitializes the Client for restarting the DHCPv6 state machine and re-running the DHCPv6 protocol. This is not necessary if the Client has not previously started the DHPCv6 state machine or been assigned any IPv6 addresses. The addresses saved to the DHCPv6 Client as well as registered with the IP instance are both cleared.

Note that the application must still start the DHCPv6 Client using the *nx_dhcpv6_start service* and begin the request for IPv6 address assignment by calling *nx_dhcpv6_request_solicit*.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Address successfully removed
NX_DHCPV6_ALREADY_STARTED	(0xE91)	DHCPv6 Client is already running
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Clear the assigned IP address(es) from the Client and the IP instance */
status = nx_dhcpv6_reinitialize(&dhcp_0);

/* If status is NX_SUCCESS the Client IP address was successfully removed.  */
```

See Also

nx_dhcpv6_stop, nx_dhcpv6_start

nx_dhcpv6_request_confirm

Process the Client's CONFIRM state

Prototype

```
UINT nx_dhcpv6_request_confirm(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service sends a CONFIRM request. If a reply is received from the Server, the DHCPv6 Client updates its lease parameters with the received data.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	CONFIRM message successfully sent and processed
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Send a CONFIRM message to the Server.  */
status = nx_dhcpv6_request_confirm(&dhcp_0);

/* If status is NX_SUCCESS the Client successfully sent the CONFIRM message.  */
```

See Also

nx_dhcpv6_request_inform_request, nx_dhcpv6_request_release, nx_dhcpv6_request_solicit

nx_dhcpv6_request_inform_request

Process the Client's INFORM REQUEST state

Prototype

```
UINT nx_dhcpv6_request_inform_request(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service sends an INFORM REQUEST message. If a reply is received, When one is received, the reply is processed to determine it is valid and the server granted the request. The Client instance is then updated with the server information as needed.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	INFORM REQUEST message successfully created and processed
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Send an INFORM REQUEST message to the server.  */
status = nx_dhcpv6_request_inform_request(&dhcp_0);

/* If status is NX_SUCCESS the Client successfully sent the INFORM REQUEST message and
    processed the reply.  */
```

See Also

nx_dhcpv6_request_confirm

nx_dhcpv6_request_option_DNS_server

Add DNS Server to DHCPv6 Option request

Prototype

```
UINT nx_dhcpv6_request_option_DNS_server(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service adds the option for requesting DNS server information to the DHCPv6 option request. If the Server reply includes DNS server data, the Client will store the DNS server if it has room to do so. The number of DNS servers the Client can store is determined by the configurable option NX_DHCPV6_NUM_DNS_SERVERS whose default value is 2.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	DNS server option is included
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Set the DNS server option in Client requests.  */
nx_dhcpv6_request_option_DNS_server(&dhcp_0, NX_TRUE);
```

See Also

nx_dhcpv6_request_option_domain_name, nx_dhcpv6_request_option_time_server, nx_dhcpv6_request_option_timezone

nx_dhcpv6_request_option_FQDN

Add Fully Qualified Domain Name option to Option request list

Prototype

```
UINT nx_dhcpv6_request_option_FQDN(NX_DHCPV6 *dhcpv6_ptr, UCHAR *domain_name,
UINT op)
```

Description

This service adds the option for adding the Client Fully Qualified Domain Name to the DHCPv6 option request. There are three options for the FQDN option:

NX_DHCPV6_CLIENT_DESIRES_UPDATE_AAAA_RR	0 Update the FQDN-to-IPv6 address mapping for FQDN and address(es) used by the Client.
NX_DHCPV6_CLIENT_DESIRES_SERVER_DO_DNS_UPDATE	1 Update the FQDN-to-IPv6 address mapping for FQDN and address(es) used by the Client to the server.
NX_DHCPV6_CLIENT_DESIRES_NO_SERVER_DNS_UPDATE	2 Request the server perform no DNS updates on the Client's behalf.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
domain_name	String holding the domain name
op	Type of FQDN option to apply (see list above)

Return Values

NX_SUCCESS	(0x00)	FQDN option is included
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Set the FQDN option in Client DHCPv6 request.  */
nx_dhcpv6_request_option_FQDN(&dhcp_0, "DHCPv6_Client",
                              NX_DHCPV6_CLIENT_DESIRES_NO_SERVER_DNS_UPDATE);
```

See Also

nx_dhcpv6_request_option_domain_name, nx_dhcpv6_request_option_time_server, nx_dhcpv6_request_option_timezone

nx_dhcpv6_request_option_domain_name

Add domain name option to DHCPv6 option request

Prototype

```
UINT nx_dhcpv6_request_option_domain_name(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service adds the domain name option to the option request in Client request messages. If the Server reply includes domain name data, the Client will store the domain name information if the size of the domain name is within the buffer size for holding the domain name. This buffer size is a configurable option (NX_DHCPV6_DOMAIN_NAME_BUFFER_SIZE) with a default value of 30 bytes.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Domain name option set
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Set the domain name option in Client requests.  */
nx_dhcpv6_request_option_domain_name(&dhcp_0, NX_TRUE);
```

See Also

nx_dhcpv6_request_option_DNS_server, nx_dhcpv6_request_option_time_server, nx_dhcpv6_request_option_timezone

nx_dhcpv6_request_option_time_server

Set time server data as optional request

Prototype

```
UINT nx_dhcpv6_request_option_time_server(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service adds the option for time server information to the option request of Client request messages. If the Server reply includes tim server data, the Client will store the time server if it has room to do so. The number of time servers the Client can store is determined by the configurable option NX_DHCPV6_NUM_TIME _SERVERS whose default value is 1.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Time server option added
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Set the time server option in Client request messages.  */
nx_dhcpv6_request_option_time_server(&dhcp_0, NX_TRUE);
```

See Also

nx_dhcpv6_request_option_DNS_server, nx_dhcpv6_request_option_domain_name, nx_dhcpv6_request_option_timezone

nx_dhcpv6_request_option_timezone

Set time zone data as optional request

Prototype

```
UINT nx_dhcpv6_request_option_timezone(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service adds the option for requesting time zone information to the Client option request. If the Server reply includes time zone data, the Client will store the time zone information if the size of the time zone is within the buffer size for holding the time zone. This buffer size is a configurable option (NX_DHCPV6_ TIME_ZONE _BUFFER_SIZE) with a default value of 10 bytes.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Time zone option added
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Set time zone option in Client request messages.  */
nx_dhcpv6_request_option_timezone(&dhcp_0, NX_TRUE);
```

See Also

nx_dhcpv6_request_option_DNS_server, nx_dhcpv6_request_option_domain_name, nx_dhcpv6_request_option_time_server

nx_dhcpv6_request_release

Send a DHCPv6 RELEASE message

Prototype

```
UINT nx_dhcpv6_request_release(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service sends a RELEASE message on the Client network. If the message is successfully sent, a successful status is returned. A successful completion does not mean the Client received a response or has been granted an IPv6 address yet. The DHCPv6 Client thread task waits for a reply from a DHCPv6 Server. If one is received, it checks the reply is valid and stores the data to the Client record.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	RELEASE message successfully sent
NX_DHCPV6_NOT_STARTED	(0xE92)	DHCPv6 Client task not started
NX_DHCPV6_IA_ADDRESS_NOT_VALID	(0xEAD)	Address not bound to Client
NX_INVALID_INTERFACE	(0x4C)	Not found in IP address table
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Send an RELEASE message to the Server.  */
status = nx_dhcpv6_request_release(&dhcp_0);

/* If status is NX_SUCCESS the Client successfully sent the RELEASE message.  */
```

See Also

nx_dhcpv6_request_confirm, nx_dhcpv6_request_inform_request, nx_dhcpv6_request_solicit

nx_dhcpv6_request_solicit

Send a SOLICIT message

Prototype

```
UINT nx_dhcpv6_request_solicit(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service sends a SOLICIT message out on the network. If the message is successfully sent, a successful status is returned. A successful completion does not mean the Client received a response or has been granted an IPv6 address yet. The DHCPv6 Client thread task waits for a reply (an ADVERTISE message) from a DHCPv6 Server. If one is received, it checks the reply is valid, stores the data to the Client record and promotes the Client to the REQUEST state.

Note that if the Rapid Commit option is set, the DHCPv6 Client will go directly to the Bound state if it receives a valid Server ADVERTISE message. See the service description for *nx_dhcpv6_request_solicit_rapid* for more details.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	SOLICIT message successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Send an SOLICIT message to the server.  */
status = nx_dhcpv6_request_solicit(&dhcp_0);

/* If status is NX_SUCCESS the Client successfully sent the SOLICIT message.  */
```

nx_dhcpv6_request_solicit_rapid

Send a SOLICIT message with the Rapid Commit option

Prototype

```
UINT nx_dhcpv6_request_solicit_rapid(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service sends a SOLICIT message out on the network with the Rapid Commit option set. If the message is successfully sent, a successful status is returned. A successful completion does not mean the Client received a response or has been granted an IPv6 address yet. The DHCPv6 Client thread task waits for a reply (an ADVERTISE message) from a DHCPv6 Server. If one is received, it checks the reply is valid, stores the data to the Client record and promotes the Client to the BOUND state.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	SOLICIT message successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Send an SOLICIT message to the server.  */
status = nx_dhcpv6_request_solicit_rapid(&dhcp_0);

/* If status is NX_SUCCESS the Client successfully sent the SOLICIT message.  */
```

See Also

nx_dhcpv6_request_solicit, nx_dhcpv6_request_confirm, nx_dhcpv6_request_inform_request, nx_dhcpv6_request_release

nx_dhcpv6_resume

Resume DHCPv6 Client task

Prototype

```
UINT nx_dhcpv6_resume(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service resumes the DHCPv6 Client thread task. The current DHCPv6 Client state will be processed (e.g. Bound, Solicit)

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client successfully resumed
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Resume the DHCPv6 Client task.  */
status = nx_dhcpv6_resume(&dhcp_0);

/* If status is NX_SUCCESS the Client thread task successfully resumed.  */
```

See Also

nx_dhcpv6_start, nx_dhcpv6_stop, nx_dhcpv6_suspend

nx_dhcpv6_set_time_accrued

Sets time accrued on Client's IP address lease

Prototype

```
UINT nx_dhcpv6_set_time_accrued(NX_DHCPV6 *dhcpv6_ptr,
                                ULONG time_accrued)
```

Description

This service sets the time accrued on the Client's global IP address since it was assigned by the server. This should only be used if a Client is currently bound to an assigned IPv6 address.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
time_accrued	Time accrued in IP lease

Return Values

NX_SUCCESS	(0x00)	Time accrued successfully set
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Set time accrued since client's assigned IP address was assigned.  */
status = nx_dhcpv6_set_time_accrued(&dhcp_0, time_accrued);

/* If status is NX_SUCCESS the time accrued on the client IP address lease was
    successfully set.  */
```

See Also

nx_dhcpv6_get_IP_address, nx_dhcpv6_get_other_option_data, nx_dhcpv6_get_lease_time_data, nx_dhcpv6_get_time_accrued

nx_dhcpv6_start

Start the DHCPv6 Client task

Prototype

```
UINT nx_dhcpv6_start(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service starts the DHCPv6 Client task and prepares the Client for running the DHCPv6 protocol. It verifies the Client instance has sufficient information (such as a Client DUID), creates and binds the UDP socket for sending and receiving DHCPv6 messages and activates timers for keeping track of session time and when the current IPv6 lease expires.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client successfully started
NX_DHCPV6_MISSING_REQUIRED_OPTIONS	(0xEA9)	Client missing required options
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Start the DHCPv6 Client task.  */
status = nx_dhcpv6_start(&dhcp_0);

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

See Also

nx_dhcpv6_resume, nx_dhcpv6_suspend, nx_dhcpv6_stop, nx_dhcpv6_reinitialize

nx_dhcpv6_stop

Stop the DHCPv6 Client task

Prototype

```
UINT nx_dhcpv6_stop(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service stops the DHCPv6 Client task, and clears retransmission counts, maximum retransmission intervals, deactivates the session and lease expiration timers, and unbinds the DHCPv6 Client socket port. To restart the Client, one must first stop and optionally reinitialize the Client before starting another session with any DHCPv6 server. See the Small Example section for more details.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client successfully stopped
NX_DHCPV6_NOT_STARTED	(0xE92)	Client thread not started
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Stop the DHCPv6 Client task.  */
status = nx_dhcpv6_start(&dhcp_0);

/* If status is NX_SUCCESS the Client successfully stopped.  */
```

See Also

nx_dhcpv6_resume, nx_dhcpv6_suspend, nx_dhcpv6_reinitialize, nx_dhcpv6_start

nx_dhcpv6_suspend

Suspend the DHCPv6 Client task

Prototype

```
UINT nx_dhcpv6_suspend(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service suspends the DHCPv6 client task and any request it was in the middle of processing. Timers are deactivated and the Client state is set to non-running.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client successfully suspended
NX_DHCPV6_NOT_STARTED	(0XE92)	Client not running so cannot be suspended
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Must be called from thread

Allowed From

Threads

Example

```
/* Suspend the DHCPv6 Client task.  */
status = nx_dhcpv6_suspend(&dhcp_0);

/* If status is NX_SUCCESS the Client successfully suspended.  */
```

See Also

nx_dhcpv6_resume, nx_dhcpv6_start, nx_dhcpv6_reinitialize, nx_dhcpv6_stop

Appendix A - Description of the Restore State Feature

The NetX Duo DHDPv6 Client configuration option, NX_DHCPV6_CLIENT_RESTORE_STATE, allows a system to restore a previously created DHCP Client in a Bound state between system reboots.

This option also allows an application to suspend the DHCPv6 Client thread and resume it, updated with the elapsed time between suspending and resuming the thread without powering down.

Restoring the DHCPv6 Client between Reboots

To restore a DHCPv6 Client between reboots, the DHCPv6 application creates an instance of the DHCPv6 Client, and then obtains an IP address lease using the normal DHCPv6 protocol and calling *nx_dhcpv6_start*. Then the DHCPv6 application waits for the protocol to complete. If all goes well, the device achieves the BOUND state with an assigned valid IP address from its DHCPv6 Server. Before it powers down, the DHCPv6 Client application saves the current DHCPv6 Client instance to a DHCPv6 Client record which is then stored in non-volatile memory. An independent 'time keeper' elsewhere in the system keeps track of the time elapsed during this powered down state. On powering up, the application creates a new DHCPv6 Client instance, and then updates it with the previously created DHCPv6 Client record. The elapsed time is obtained from the "time keeper" and then applied to the time remaining on the DHCP Clientv6 lease. At this point, the application can resume the DHCPv6 Client.

If the time elapsed during power down puts the DHCPv6 Client state in either a RENEW or REBIND state, the DHCPv6 Client will automatically initiate DHCPv6 messages requesting to renew or rebind the IP address lease. If the IP address is expired, the DHCPv6 Client will automatically clear the IP address on the IP instance and begin the DHCPv6 process from the INIT state, requesting a new IP address.

In this manner the DHCPv6 Client can operate between reboots as if uninterrupted.

Below is an illustration of this feature.

```
/* On the power up, create an IP instance, DHCPv6 Client, enable ICMPv6 and UDP
   and other resources (not shown) for the DHCPv6 Client/application
   in tx_application_define().  */


/* Define the DHCPv6 Client application thread. */
void    thread_dhcpv6_client_entry(ULONG thread_input)
{

UINT        status;
UINT        time_elapsed = 0;
NX_DHCPV6_CLIENT_RECORD client_my_record;


    /* No previously saved Client record. Start the DHCPv6 Client in the INIT state.  */
    status =  nx_dhcpv6_start(&dhcp_0);

    if (status !=NX_SUCCESS)
        return;
```

```
    while(1)
    {

        /* Wait for DHCPv6 Client to get the IP address.  */
    }

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

    /* Save the Client state data which we will subsequently need to restore the DHCPv6
       Client. */
    status = nx_dhcpv6_client_get_record(&dhcp_0, &client_my_record);

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

    /* Delete the IP and DHCPv6 Client instances before powering down. */
    nx_dhcpv6_client_delete(&dhcp_0);

    nx_ip_delete(&ip_0);

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

    /* The application has determined there is a previously saved record. We will
       restore it to the current DHCPv6 Client instance.  */

    /* Create the IP and DHCPv6 Client instances, enable ICMPv6 and UDP after powering up.  */

    /* Calculate the time elapsed during power down */

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

    /* Apply the record to the current Client instance. This will also
       update the IP instance with IP address, mask etc. */
    status = nx_dhcpv6_client_restore_record(&dhcp_0, &client_my_record, time_elapsed);

     if (status != NX_SUCCESS)
          return;

     /* We are ready to resume the DHCPv6 Client thread and use the assigned IP address. */
     status = nx_dhcpv6_resume(&dhcp_0);

     if (status != NX_SUCCESS)
          return;

}
```

nx_dhcpv6_client_get_record

Create a record of the current DHCPv6 Client state

Prototype

```
ULONG nx_dhcpv6_client_get_record(NX_DHCPV6 *dhcpv6_ptr,
                                  NX_DHCPV6_CLIENT_RECORD *record_ptr);
```

Description

This service saves the DHCPv6 Client to the record pointed to by `record_ptr`. This allows the DHCPv6 Client application restore its DHCPv6 Client state after, for example, a power down and reboot.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client
record_ptr	Pointer to DHCPv6 Client record

Return Values

NX_SUCCESS	(0x0)	Valid Client record created
NX_DHCPV6_NOT_BOUND	(0xE94)	Client not in bound state, therefore not assigned valid IP address
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
NX_DHCPV6_CLIENT_RECORD dhcpv6_record;


/* Obtain a record of the current client state. */
status=  nx_dhcpv6_client_get_record(&dhcpv6_ptr, &dhcpv6_record);

/* If status is NX_SUCCESS dhcpv6_record contains the current DHCPv6 client record. */
```

See Also

nx_dhcpv6_resume, nx_dhcpv6_suspend, nx_dhcpv6_client_restore_record

nx_dhcpv6_client_restore_record

Restore DHCPv6 Client state from saved record

Prototype

```
ULONG nx_dhcpv6_client_restore_record(NX_DHCPV6 *dhcpv6_ptr,
                                      NX_DHCPV6_CLIENT_RECORD
                                      *record_ptr, ULONG time_elapsed);
```

Description

This service enables a DHCPv6 application to recreate its DHCPv6 Client state from a previous session by updating the DHCPv6 Client with the DHCPv6 Client record pointed to by `record_ptr`, and updates the time remaining on DHCPv6 Client lease with the `time_elapsed` input. This allows the DHCPv6 Client application to recreate its DHCPv6 Client, for example, after powering down. This requires that the DHCPv6 Client application created a record of the DHCPv6 Client before powering down, and saved that record to non-volatile memory.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client
record_ptr	Pointer to DHCPv6 Client record
time_elapsed	Time to subtract from the lease time remaining in the input client record

Return Values

NX_SUCCESS	(0x0)	Client record restored
NX_PTR_ERROR	(0x16)	Invalid Pointer Input

Allowed From

Threads

Example

```
NX_DHCPV6_CLIENT_RECORD dhcpv6_record;
ULONG          time_elapsed;

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

/* Obtain a record of the current client state. */
status=  nx_dhcpv6_client_restore_record(&dhcpv6_ptr, &dhcpv6_record, time_elapsed);

/* If status is NX_SUCCESS the current DHCPv6 Client pointed to by dhcpv6_ptr contains
the current client record updated for time elapsed during power down. */
```

See Also

nx_dhcpv6_client_get_record