Dynamic Host Configuration Protocol for IPv6 Clients (DHCPv6 Client)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

©2002-2013 by Express Logic, Inc.
All rights reserved. This document and the associated NetX Duo software are the sole property of Express Logic, Inc. Each contains proprietary information of Express Logic, Inc. Reproduction or duplication by any means of any portion of this document without the prior written consent of Express Logic, Inc. is expressly forbidden.
Express Logic, Inc. reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of NetX. The information in this document has been carefully checked for accuracy; however, Express Logic, Inc. makes no warranty pertaining to the correctness of this document.

Trademarks
NetX, Piconet, and UDP Fast Path are trademarks of Express Logic, Inc. ThreadX is a registered trademark of Express Logic, Inc.
All other product and company names are trademarks or registered trademarks of their respective holders.

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

Part Number: 000-1050
Revision 5.4

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 preprocesses 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 (the 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 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.

NetX Duo DHCPv6 Client Requirements and Constraints

The NetX Duo DHCPv6 Client services require NetX Duo 5.6 or later.

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.

Client DUIDs

A Client DHCPv6 Unique Identifier (DUID) uniquely defines each device on a network. Before an application starts the DHCPv6 Client, the application must call the *nx_dhcpv6_create_client_duid* service to create a Client DUID which must be in all its messages to the Server:

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

The Client link layer address is needed to create the same Client DUID of type link layer and link layer plus time. The link layer need not require non volatile memory if the Client has access to the link layer address. For DUIDs of type time, the same time data must be used in the DUID creation and this does require non volatile memory. Clients that do not have any stable storage must not use DUIDs of type time.

IANA and IA Addresses

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

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. Before starting the DHCPv6 Client it creates the 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)
```

The Client must also save the IPv6 address in the IA blocks to non volatile memory. Before starting the DHCPv6 Client, the Client creates the IA using the *nx_dhcpv6_create_client_ia* service and saved 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)
```

Lease Expiration Time

The Client must 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 Client 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.

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 Fully Qualified Domain Name (FQDN) option.

- 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.

Setting Up the DHCPv6 Client

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, and can do so by calling 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 using a previously assigned IPv6 address lease, see “Non Volatile Memory Requirements” in the previous chapter for creating the Client DUID.

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. 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 shown below and demonstrated in the “Small Example System” elsewhere in this chapter:

```
UINT    nx_dhcpv6_create_client_iana(NX_DHCPV6 *dhcpv6_ptr,
                                     UINT IA_ident, ULONG T1, ULONG T2)
UINT    nx_dhcpv6_add_client_ia(NX_DHCPV6 *dhcpv6_ptr,
                                NXD_ADDRESS *ipv6_address,
                                ULONG preferred_lifetime,
                                ULONG valid_lifetime)
```

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)
```

These services are described in greater detail in Chapter 4 and demonstrated in the “Small Example System” section that follows.

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.*

Note: If the application wishes to use a previously assigned IPv6 lease assigned, it calls *nx_dhcpv6_start* with a non zero time input. This is discussed in more detail in “Non Volatile Memory Requirements” in Chapter 1. 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 the IP instance. Then the DHCPv6 Client is created with state change (*dhcpv6_state_change_notify*) and server error (*dhcpv6_server_error_handler*) callback functions.

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. Before starting the DHCPv6 Client, the application creates a Client DUID, an IANA option and an IA address option. 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*, and the actual DHCPv6 protocol is started in the SOLICIT state with the call to *nx_dhcpv6_request_solicit*. 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.* 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*.

The application then releases the assigned addresses using the *nx_dhcpv6_request_release* service. To restart the application stops the DHCPv6 Client with the *nx_dhcpv6_client_stop* service and clears all IPv6 addresses registered with the IP instance that were configured throughthe DHCPv6 Client. It does this by calling *nx_dhcpv6_reinitialize*. 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*. 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.

[YZ: see demot_netxduo_dhcpv6_client.c for actual demo]

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.

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
(*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_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_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_clinet_delete(&my_dhcp);

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

See Also

nx_dhcpv6_client_create

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_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_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_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_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 has successfully set the DHCPv6 network interface.
    */
```

See Also

nx_dhcpv6_client _create, nx_dhcpv6_start

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(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_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_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