Dynamic Host Configuration Protocol over IPv6 (DHCPv6)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

©2002-2012 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 5.3

Contents

Chapter 1

Introduction to DHCPv6 Client

Before the development of IPv6 and DHCPv6, aNetX Duo host's IP address was one of the supplied parameters to the *nx_ip_create* service call in NetX Duo based applications. Supplying the IP address posed no problem if the IP address was known to the host, either statically or through user configuration. However, there were some instances where the host didn't know or care what its IP address was. In such situations, a zero IP address would be supplied to the *nx_ip_create* function and the DHCP Client protocol would be used to dynamically obtain an IP address.

In IPv6 networks, DHCPv6replaces DHCP and offers most of the same features as well as many enhancements. ARP has been replaced by IPv6 Neighbor Discovery protocol. AClient who does not or cannot use IPv6 stateless address auto-configuration must use DHCPv6 to be assigned a unique global IP address from a DHCPv6 server.

NetX Duo was developed by ExpressLogic to support IPv6 network based applications and network protocols such as DHCPv6. This document will explain in detail how the NetX Duo DHCPv6 Client API can be used to set up IPv6 address assignment for host applications.

DHCPv6 Communication

DHCPv6 clients and servers exchange messages using UDP. The Client uses port 546 to send and receive DHCPv6 messages and the server uses port 547. The Client initially uses its link-local address for transmitting and receiving DHCP messages. It sends all messages to DHCPv6 servers using a reserved,link-scoped multicast address known as the *All_DHCP_Relay_Agents_and_Servers* multicast address *FF02::01:02***.** The Clientuses this address to send DHCPv6 messages regardless if the Clientdesires to reach all servers or just one. Therefore, the Clientneed not be configured with the address of DHCPv6 server(s) on its link.

To request assignment of a global IPv6 address, a Clientfirst locates a DHCPv6 server. It does this by sending a *Solicit* message with the reserved multicast address for DHCPv6 servers. In the *Solicit* request, the Clientcan request the assignment of specific IPv6 address or let the server choose one. It can also request other network configuration information from the server.

A DHCPv6 Server that can service a Client request responds with an *Advertise* message containing the IPv6 address it will grant to the Client, the IPv6 address lease time and any additional information requested by the Client. Normally, a DHCPv6 Client waits a period of time to receive Advertise messages from all DHCPv6 Servers on the network, before deciding which one it would prefer to or in some cases must respond to. However, the NetX Duo DHPv6 Client chooses the first valid server reply it receives. The Client extracts server data and determines that the server reply is indeed valid so the Client can accept what the server will offer.

Next the Client 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 is registered with the server and assigned to the client.

In most situations, more information than an IPv6 address is necessary for a Client to properly participate on a network. In addition to an IPv6 address, most clients need the IPv6 address of DNS server(s) or time server(s) on the link, as well as other network information such as the network domain name. DHCPv6 has the ability to provide this information using *Information Request* messages which are explained later in this Chapter.

When the DHCPv6 Client IPv6 address lease nears the preferred life time timeout, the DHCPv6 Client signals the host application via the address deprecation handler, if the DHCPv6 Client *nx_dhcpv6_deprecated_IP_address_handler* is defined. When the IPv6 address reaches the address expiration timeout, the DHCPv6 Client signals the host application via the address expiration handler, if the DHCPv6 Client *nx_dhcpv6_expired_IP_address_handler* is defined. The host application should attempt to renew or rebind its address respectively. See the Small Example section for details.

The *Renew* and other DHCPv6 message types are explained in detail in the next section.

NetX Duo DHCPv6 Client Requirements and Constraints

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

IP Thread Task Requirements

The NetX Duo DHCPv6 Client API requires a creation of a NetX Duo IP instance for sending and receiving messages to DHCPv6(s) on its network link.

DHCPv6 utilizes NetX Duo, ICMP and UDP, so IPv6 must be enabled prior to using DHCPv6:

- *nxd_ipv6_enable*
- *nx_udp_enable*
- *nx_icmp_enable (optional)*

Packet Pool Requirements

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

A typical DHCPv6 message is about 200-300 bytes depending on the number of DHCPv6 options requested by the Client.

Other Requirements

The NetX Duo DHCPv6 Client requires the creation of UDP socket bound to port 546. This is handled by internal processing when the DHCPv6 Client task is started.

The host application MUST save certain DHCPv6 Client data to non volatile memory so that it can present the same data to DHCPv6 Servers across multiple reboots as explained below. See the “Small Example System” elsewhere in this document for a demonstration:

A Client DHCPv6 Unique Identifier (DUID) uniquely defines each Client host on a network. This data must be consistent across Client host reboots. Before the host application starts the DHCPv6 Client on rebooting, the host application should call the *nx_dhcpv6_create_client_duid*service using information saved to non volatile memory to restore its DUID.

A Client Identity Association for Non Temporary Addresses (IANA) and Identity Association IPv6 address (IA) cumulatively define theClient IPv6 address assignment parameters:

- The Clienthost must save the IANA parameters (T1, T2, IANA identifier) tonon volatile memory so that on rebooting it can restore

the IANA before starting the DHCPv6 Client using the *nx_dhcpv6_create_client_iana* service

- The Client host must also save its assigned IPv6 address.When rebooting, the Client can restore its IA using the *nx_dhcpv6_create_client_ia* service before starting the DHCPv6 Client.

- Finally, the Client host must store the accrued time on its IPv6 address lease to non volatile memory if shutting down. Further the host application must have an independent clock to update the accrued time for the interval during which the device is shut down. When it restarts the Client task with the *nx_dhcpv6_start*, it includes the accrued time as one of the inputs. From here on, the Client task will handle a lease approaching or passed expiration.

Limitations

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

- NetX Duo DHCPv6 Client accepts the first valid DHCPv6 Server Advertisement received in response to its Solicit message, regardless of the Server “Preference” level in the Advertisement. DHCPv6 protocol recommends the Client to wait a specified time to receive responses from all Serverson the network responses then choose among them based on Preference level as well as IP address and options offered.

- NetX Duo DHCPv6 Client does not support multiple IPv6 address assignment which the DHCPv6 protocol allows for. The NetX Duo DHCPv6 Client only maintains a single IPv6 address in a single IANA control block.

- NetX Duo DHCPv6 Client does not support unicast messages to the DHCPv6 Server even if the Server indicates this is permitted. The DHCPv6 protocol allows for unicast messaging to reduce bandwidth required by multicast messaging.

- NetX Duo DHCPv6 Client does not support the Rapid Commit option which allow Client and Server to conclude IPv6 address assignment in just two messages, instead of four.

- NetX Duo DHCPv6 Client does not support the Reconfigure request in which aServer 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 PlusTime format.

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

Multihome support for the DHCPv6 Client

The DHCPv6 Client release 5.1 or later supports multihome devices, and introduces a new service, *nx_dhcpv6_client_set_interface,* for setting the network interface which the host application will be communicating with the DHCPv6 Server. Note: this option defaults to the primary interface (index zero) for DHCPv6 Clients on single home devices or hosts on NetX Duo environments that do not support multihoming. See Chapter 4 for details on this service.

Setting Up the DHCPv6 Client

Client DHCP Unique Identifier (DUID)

The Client DUID uniquely defines each Client host on a network. If a Client does not have a DUID, it can use the NetX Duo DHCPv6 Client Service*nx_dhcpv6_create_client_duid*to create one.

The Client host application provides all the DUID parameters except for the time field in the Link Layer Plus Time type when it creates a Client DUID. The time input is optional: if the host application does not provide this data, the NetX Duo DHCPv6 Client providesone. See Chapter 4 for the details of these API services.

Identity Association for Non Temporary Addresses

The Identity Association for Non Temporary Addresses (IANA) and Identity Association IPv6 address (IA) in the NetX Duo DHCPv6 Client record define the Client IPv6 address assignment and lease parameters. To create an IANA or IA Address option, the host application can call the *nx_dhcpv6_create_client_iana* and *nx_dhcpv6_create_client_ia* services to create them. The Client is strongly recommended to save the IANA and IA data to nonvolatile memory so as to be able to retrieve it after the application is terminated or the host is powered down. The application host is strongly recommended to save assigned IPv6 address and expiration timeouts across reboots as well.

The following services allow the host application to create the IANA and IA for use in requesting an IP address from a DHCPv6 server:

nx_dhcpv6_create_client_iana
nx_dhcpv6_create_client_ia

The host can request a specific IPv6 address and lease lifetime. When an IPv6 lease is assigned, it will update its IANA and IA with the IP address and lease times assigned by the Server:

The following services allow the host application to retrieve pertinent data from the IANA and IA address control block after the NetX Duo DHCPv6 Client has obtained a valid IPv6 address from the Server:

nx_dhcpv6_get_lease_time_data
nx_dhcpv6_get_IP_address

The host application starts the DHCPv6 Client thread by calling the *nx_dhcpv6_start* service. If the Client has not been assigned an IPv6 lease, it should set the time argument to zero when calling this service. Otherwise it should use the time expired since the lease was assigned. This may require access to a real time clock or other external component while the Client is stopped or the host is shut down.

Here is sample code demonstrating how to start and resume the DHCPv6 Client task:

```
NX_DHCPV6   dhcpv6_client;
UINT nv_time;

nv_time = 0;
nx_dhcpv6_start(&dhcpv6_client, nv_time);

/* Begin the process of requesting an IPv6 lease with the
   SOLICIT request. The DHCPv6 Client will complete this
   protocol for the host. */
nx_dhcpv6_request_solicit(&dhcpv6_client);
```

To resume the DHCPv6 Client:

```
NX_DHCPV6   dhcpv6_client;
UINT nv_time;

nv_time = [time elapsed since IP lease assigned];
nx_dhcpv6_start(&dhcpv6_client, nv_time);

/* Optional: host should check with the DHCPv6 Server that
   its IPv6 lease still valid */
nx_dhcpv6_request_confirm((&dhcpv6_client);
```

See the “Small Example System” in Chapter 2 or list of services in Chapter 4 for more information about these API services.

When the IPv6 address is assigned to the Client, the Clientis responsibleforverifying that the address is unique on the network. When it registers the assigned global address with NetX Duo, as demonstrated below, NetX Duo will automatically validate the address if Duplicate Address Detection (DAD) is enabled. See the NetX Duo User Guide for more details on DAD.

```
    /* Now let NetX Duo know about our global IP address. This assumes
the assignedIPv6 lease address has a 64 bit prefix and is
       assigned for the primary interface (0). */

    /* Set the global address at index 1 in the IP address table.*/
ga_address_index = 1;

    nxd_ipv6_address_set(&ip_0,0,&ipv6_address,64,&ga_address_index);

    /* Give Netx Duo and Duplicate Address Detection (if enabled) a
chance to validate the IPv6 address. This takes about 3-4
seconds. */
    tx_thread_sleep(400);
```

Note that if the address is already in use or the host application does not want to accept the assigned IP address, it may decline the address using the *nx_dhcpv6_send_decline* service. This is described in greater detail later in this chapter.

IPv6 Address Lease

When a Client is assigned an IPv6 address from a DHCPv6 server, this address usually has a limited lease time. DHCPv6 protocol includes a series of time expirations at which time the status of the Client address changes.

The DHCPv6 Client task keeps track of time elapsed from when an IPv6 address is assigned. It will detect when any of the times above have expired. When one does, it will automatically trigger the appropriate NetX Duo DHCPv6 Client response and send the corresponding (e.g. renew, rebind) message to the server. The NetX Duo DHCPv6 Client has two callback functions for the host application to be notified if the Client IPv6 address has become deprecated or expired. These are discussed in greater detail in the section **NetX Duo DHCPv6 Client Callback Functions** later in this chapter:

nx_dhcpv6_deprecated_IP_address_handler
nx_dhcpv6_expired_IP_address_handler

The Client can indicate preferred lease times it would like to be assigned when it submits its IP address requests to the server. To receive the maximum lease length, the client sets these fields to infinity (0xFFFFFFFF). To let the server choose, it sets them to zero.

The *IANA* control block contains the T1 and T2 fields. The *IA* block in the *IANA* control block contains the preferred and valid lifetime fields. The countdown on these time expirations begins as soon as the Client receives confirmation (server *Reply*) to its *Solicit*, *Renew* or *Rebind* requests for its address assignment.

These fields, and the action the NetX Duo DHCPv6 Client task automatically initiates, are defined below.

T1 – time in seconds at which the Client must renew its IPv6 address from the same server that assigned it. The NetX Duo DHCPv6 Client automatically sends a *Renew* message to the server.

T2 – time in seconds at which if the Client was unable to renew its IP address and must rebind the IPv6 address with any server on its link. The NetX Duo DHCPv6 client automatically sends a *Rebind* message to the server.

Preferred lifetime – time in seconds when the Client address has become deprecated. The Client may still use this address, but it is a deprecated address. The NetX Duo DHCPv6 Client automatically calls the *nx_dhcpv6_deprecated_IP_address_handler* if one is defined for the Client.

Valid lifetime –time in seconds when the Client IP address is expired and must not use this address in its network transmissions. The NetX Duo DHCPv6 Client automatically calls the *nx_dhcpv6_expired_IP_address_handler*if one is defined for the Client.

The RFC recommends T1 and T2 times that are 0.5 and 0.8 respectively of the preferred lifetime in the *IANA* control block. If the host application has no preference, it should set these times to zero. If the Server reply contains T1 and T2 times set to zero, the Client may set its own T1 and T2 times and record them in its IANA.

Note! If the host application, with an assigned IP address with a finite lease time, suspends the DHCPv6 Client task and/or powers down, it must save the time elapsed before stopping the task. If shutting down, it must save the time to non volatile memory. While the Client task is not running, the host application must utilize an alternate time source, e.g. a real time clock, to continue to track time elapsed. On power up and before resuming the DHCPv6 Client, the host must

then retrieve the current time elapsed and update the Client task with this data in the *nx_dhcpv6_start* nv_time argument. It can do so using the following API:

nx_dhcpv6_get_accrued_time
nx_dhcpv6_set_accrued_time

These API are described in greater detail in Chapter 4.

DHCPv6 Messages

DHCPv6 Message structure

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

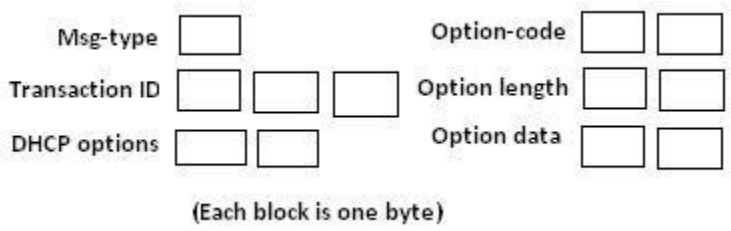

Figure 1. DHCPv6 message and option block structure

The 1-byte Msg-Type field indicates the type of DHCPv6 message. The 3-byte Transaction-ID field is determined by the Client. It can by any sequence of characters but must be unique to avoid confusion with other client hosts' messages to the Server. The same transaction ID must be included for all the messages in a DHCPv6 request. Following the Transaction-ID field, are one or more DHCPv6 options used to indicate what the Client is requesting.

The DHCPv6 option structure is composed of an option code, the length of option data, not including option code and length data, and finally the option data itself. The 2-byte Option-Code field indicates a specific option. The 2-byte Option-Len field indicates the length of the Option-Data field in bytes. The Option-Data field contains the data for the option.

Some option blocks are recursive. For example an *Identity Association for Non Temporary Address (IANA)*option contains *Identity Association (IA)* option to request an IPv6 address. The *IANA* option returned in the server reply also contains an *IA* option with the IPv6 address and lease times it is granting the

client, as well as a *Status* option indicating server success or error status with granting the request.

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

DHCPv6 Message Types

Although DHCPv6 greatly enhances the functionality of DHCP, it uses the same number of messages as DHCP and supports the same vendor options as DHCP. The list of DHCPv6messages are as follows:

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

Only RECONFIGURE is not supported in NetX Duo DHCPv6.

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

Client Solicit (discovery) → Server Advertisement (offer) → Client Request (accepts) → Server Reply (ACK)

SOLICIT: Initiating the IP address request

A Solicit message contains the options necessary for a Client to indicate it is requesting IPv6 address assignment and optionally additional network configuration information. In Figure 2 below, the Solicit message contains the Client DUID, IANA and embedded IA option, elapsed time and option request blocks. The elapsed time is simply the time (in hundredths of seconds) elapsed since the Client initiated the current exchange with the server. It is used by DHCPv6 Servers for administrative reasons and is not directly involved with DHCPv6 protocol.

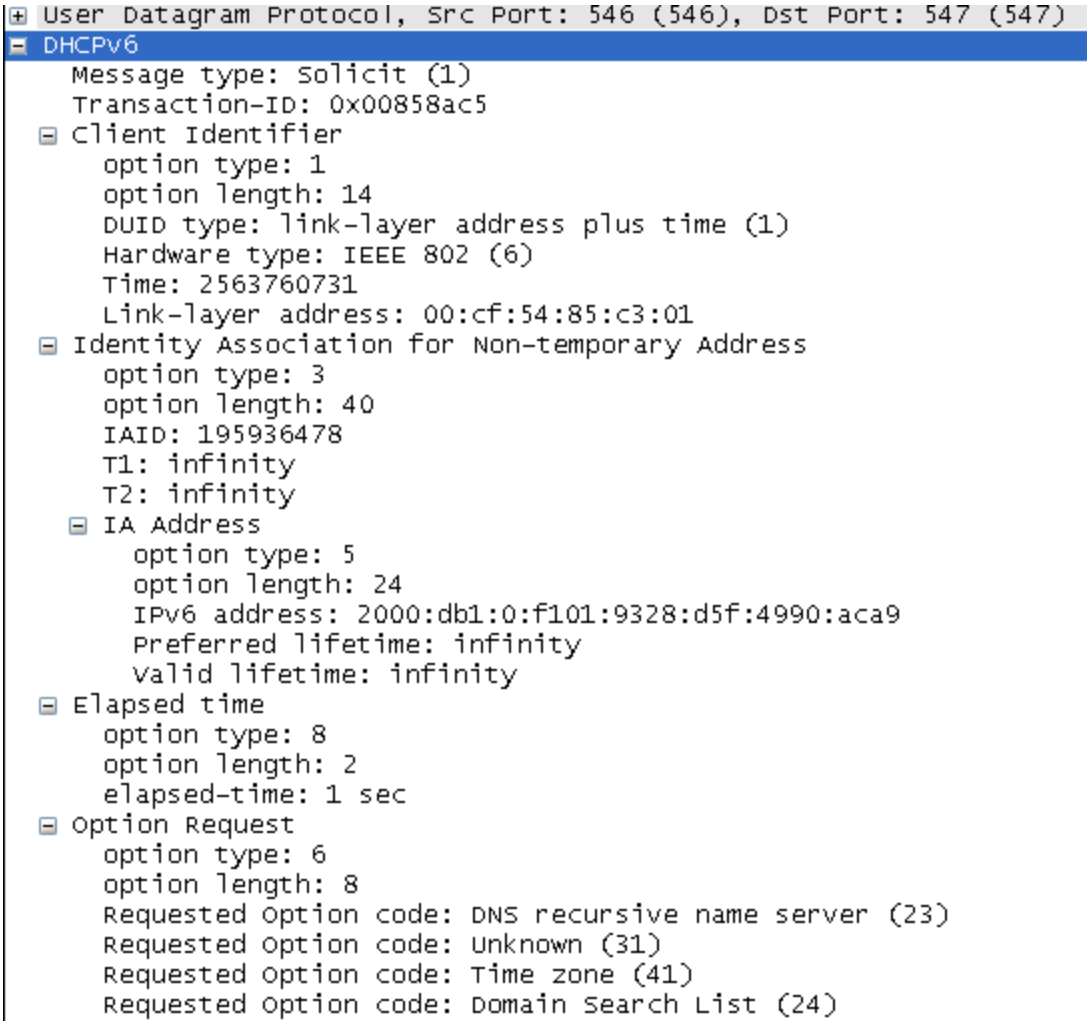

Figure 2. DHCPv6 Client Solicit message

The Client can request a specific IPv6 address or leave the field NULL in the *nx_dhcpv6_create_client_ia* API to let the Server choose the IP address. The Client can also indicate preferred T1 and T2 times which determine when the Client must renew and rebind its IPv6 address, respectively. These two actions are explained in further detail later in this section. In Figure 2, the Client has also included additional network parameters in the *Option Request* block.

A typical Server Advertisement message is shown below in Figure 3:

```
DHCPv6
  Message type: Advertise (2)
  Transaction-ID: 0x00858ac5
  Client Identifier
    option type: 1
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: IEEE 802 (6)
    Time: 2563760731
    Link-layer address: 00:cf:54:85:c3:01
  Identity Association for Non-temporary Address
    option type: 3
    option length: 74
    IAID: 195936478
    T1: 200
    T2: 300
    IA Address
      option type: 5
      option length: 24
      IPv6 address: 2000:db1:0:f101:9328:d5f:4990:aca9
      Preferred lifetime: 360
      Valid lifetime: 720
    Status code
      option type: 13
      option length: 30
      Status Code: Success (0)
      Status Message: All addresses were assigned.
  DNS recursive name server
    option type: 23
    option length: 16
    DNS servers address: 2000:db1:0:f101::101
  Domain Search List
    option type: 24
    option length: 13
    DNS Domain Search List
    Domain: example.com
  DHCP option 31
    option type: 31
    option length: 16
  Time zone
    option type: 41
    option length: 5
    time-zone: CET
  Server Identifier
    option type: 2
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: IEEE 802 (6)
    Time: 256273007
    Link-layer address: 00:19:d1:60:85:8b
  Preference
    option type: 7
    option length: 1
    pref-value: 0
```

Figure 3. DHCPv6 Server Advertisement message

In its response to the Client, the Server has filled in the T1 and T2 fields in the *IANA* option, the preferred and valid lifetime fields in the *IA* option and indicate the Server has granted the Client the IP address in the *IA* option in the *Status* option. The Server has also included its DUID for the Client to be able to identify it by. The *Preference* option which is how the Server can influence which server

the Client chooses has a value set to zero, indicating the Server is letting the Client choose its Server, which in this case is academic since there is only one DHCPv6 Server on the link.

Note the Server must include the Client's DUID in its reply message for the Client to be able to verify the reply. Rules for validating and verifying DHCPv6 messages between Client and Server are discussed later in this section.

The Server has also supplied the information request for domain name (option request type 24), time zone(option request type 41), and DNS Server (option request type 23). The NetX Duo DHCPv6 Client API attempts to filter out unprintable (non ASCII) characters in Server replies containing text.

The request for time (NTP) Server (option request type 31) is not recognized by the trace application, which is why the reply data is not displayed. However, it is included in the reply as indicated by the data length of 16 bytes.

REQUEST: Accepting the Server IP address assignment

The Client sends a *Request* message to accept the Server IPv6 address assignment in the previous *Advertisement* message. As shown in Figure 4 below, the *Request* message includes the Server's DUID. The NetX Duo DHCPv6 Client updates the *IA* option in the Client instance with the IPv6 address and valid and preferred lifetimes in the server *Advertisement IA* option.

```
DHCPv6
  Message type: Request (3)
  Transaction-ID: 0x00859ee9
  Client Identifier
    option type: 1
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: IEEE 802 (6)
    Time: 2563760731
    Link-layer address: 00:cf:54:85:c3:01
  Elapsed time
    option type: 8
    option length: 2
    elapsed-time: 21 sec
  Option Request
    option type: 6
    option length: 8
    Requested Option code: DNS recursive name server (23)
    Requested Option code: Unknown (31)
    Requested Option code: Time zone (41)
    Requested Option code: Domain Search List (24)
  Server Identifier
    option type: 2
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: IEEE 802 (6)
    Time: 256273007
    Link-layer address: 00:19:d1:60:85:8b
  Identity Association for Non-temporary Address
    option type: 3
    option length: 40
    IAID: 195936478
    T1: infinity
    T2: infinity
    IA Address
      option type: 5
      option length: 24
      IPv6 address: 2000:db1:0:f101:9328:d5f:4990:aca9
      Preferred lifetime: 120
      Valid lifetime: 150
```

Figure 4. DHCPv6 Client Request message

The Server then acknowledges the Client request with a reply message. The Client uses the server DUID and the Client DUID to verify the Server reply.

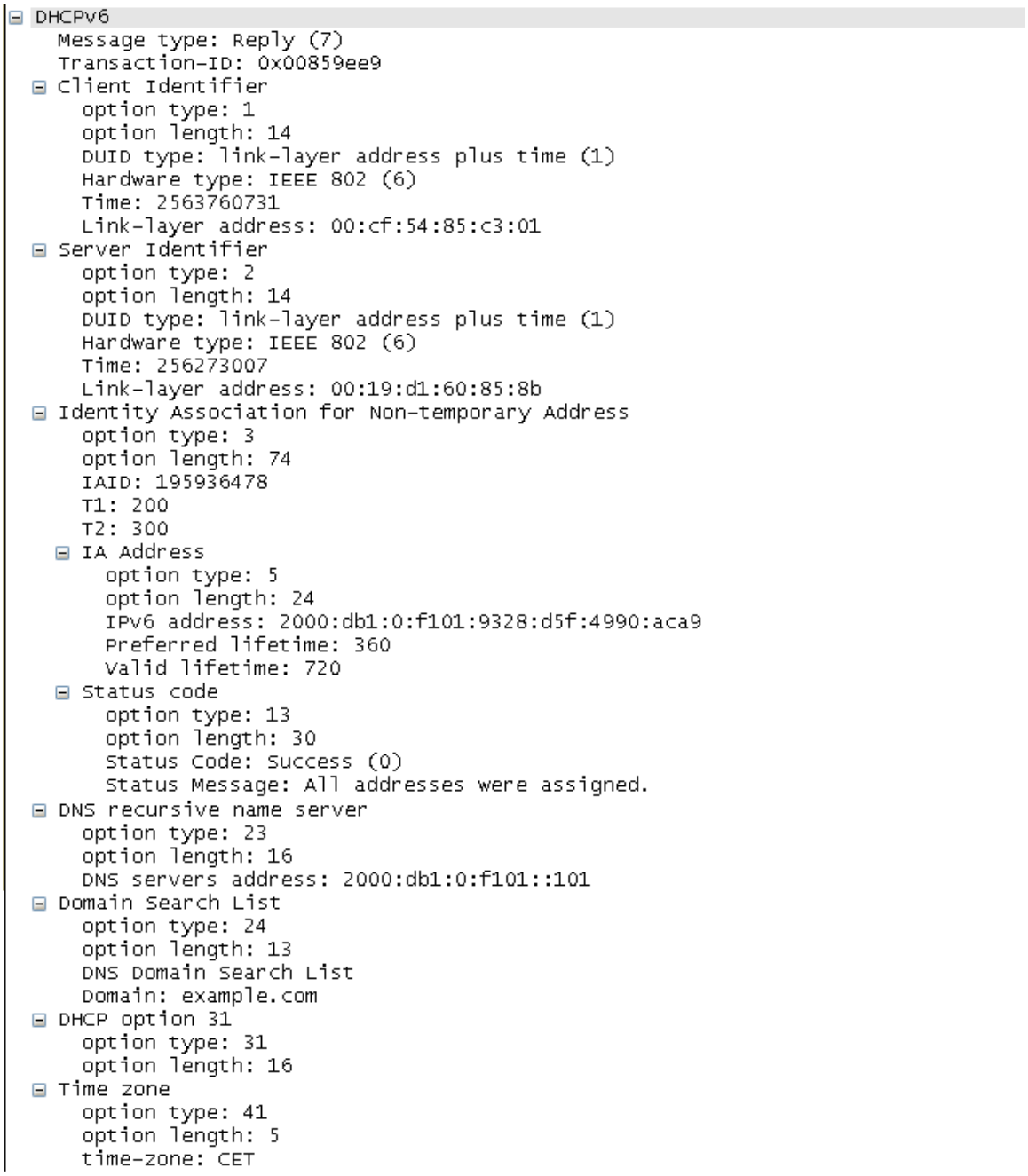

Figure 5:DHCPv6 Server Reply message

RENEW: Renewing the assigned IP address

As described previously, the NetX Duo DHCPv6 Client automatically sends a *Renew* request to the Server who assigned the IPv6 address when the T1 time

expires. A *Renew* message is identical to a *Request* message except for the message type and of course a unique message header transaction ID.

Note that the host application does not necessarily have to wait till that time, and can renew the IPv6 address lease anytime. The NetX Duo DHCPv6 Client will reset all time expirations in the Client instance if the *Renew* or *Rebind* request is successful.

The *Renew* request includes the Server DUID in the request message even though it sends the request out as a multicast packet. This is required in DHCPv6 in case of the possibility of other DHCPv6 Servers on the network.

If the Server has renewed the IPv6address, the status option in the *Reply* message in the Server reply will indicate success (0) in the data field. As with all messages from the Server, the Client verifies the Server by the Server DUID in its reply. At this point the NetX Duo DHCPv6 Client updates its IP lease with the data in the Server reply and resets the all associated time expirations to the full time out value.

REBIND: Rebinding the assigned IP address

As described previously, the NetX Duo DHCPv6 Client automatically sends a *Rebind* request to the Server who assigned the IPv6 address when the T2 time expires. T2 will only expire if the Client was unable to renew its lease when T1 expired. A *Rebind* message is identical to a *Request* message except for the message type, and it does not contain a server DUID since it is asking any Server on the link to extend its IPv6 address.

Note that the host application does not necessarily have to wait till that time, and can rebind the IPv6 address lease anytime after the renewal attempt fails.

The *Rebind* request includes the server DUID in the request message which is again sent out as a multicast packet.

If a Server has rebound the Client IPv6address, the status option in the *Reply* message in the Server reply will indicate success (0) in the data field. The Client updates any new IPv6 lease data into its IANA and IA control blocks and resets the all associated time expirations to the full time out value.

CONFIRM: Confirming an assigned IP address

A host should confirm its assigned IPv6 address with its Server if it may have moved to a new link or rebooted. The *Confirm* message includes the Server

DUID for the server that assigned its IPv6 address. The Server, unless the address has expired, should send a *Reply* message that the Client IPv6 address is still valid in a reply message. The *Status* option in the *Reply* message indicates this with success (0) in the data field. If not, the *Status* option contains an error code to let the Client know its IPv6 address is invalid. If the host application defined a Server error callback function, the NetX Duo DHCPv6 Client will notify the host application. NetX Duo DHCPv6 Client callback functions are described later in this chapter.

If the Client receives no reply to its *Confirm* request from the server, the host may still continue to use its IPv6 address. However, if the Server replies with changed IPv6 address information, the NetX Duo DHCPv6 Client will change to that IPv6 address, update its IANA and IA control blocks with changed lease time data, and update any other network information in the Server reply.

DECLINE: Declining an assigned IP address

The host application may need to decline an assigned IP address. For example, the IPv6 duplicate detection protocol may detect another host with the same IP address. To do so, the Client must send the *Decline* message as always, in a multicast packet. The *Decline* message includes the Server DUID for the Server that assigned its IP address. Before attempting any network transmissions, the host will have to initiate another IPv6 request with a *Solicit* message. The Server acknowledges the *Decline* request with a *Reply* message.

A *Decline* message looks similar to a Request message except that optional data is removed, and the message type is *Decline*.

RELEASE: Releasing an assigned IP address

If the host no longer needs or wants a previously assigned IP address, it should send a *Release* message out to the Server as always, in a multicast packet. It includes the Server DUID for the server that assigned its IP address. At this point the Client does not have to wait for the Server Reply message. NetX Duo DHCPv6 changes the Client UDP socket status to link local address and sets the global IP address to NULL.

Note: when the Client releases an IPv6 address, it is responsible for deleting the address registered with NetX Duo on its IP instance. The following service is recommended for doing so, where the global address index, *ga_address_index*, was set previously by NetX Duo when the address was registered using the *nxd_ipv6_address_set* service:

```
nxd_ipv6_address_delete(&ip_0,ga_address_index);
```

The text message in the *Status* option in all Server replies is saved to the Client record but is not processed in anyway. It has no direct effect on the DHCPv6 protocol.

INFORMATION REQUEST: Requesting optional network configuration information

The Client can request additional information in its Solicit, Renew, Confirm or Rebind messages by adding option types to the Option Request block. This is shown in the Solicit message displayed in Figure 2. The host application can use the NetX DuoDHCPv6 Client API to enable or disable options it would like to request as shown in the example below:

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

If enable is set to NX_TRUE, the option is requested. If it is set to NX_FALSE, the request is turned off. After the NetX Duo DHCPv6 Client submits the request to the Server, all options are still in effect. It is up to the host application to set or remove option requests as needed.

The complete list of option data supported by the NetX Duo DHCPv6 Client is shown below:

Option data	Code	
DNS server	23	NX_DHCPV6_DNS_SERVER_OPTION
Domain name	24	NX_DHCPV6_DOMAIN_NAME_OPTION
Time (NTP) server	31	NX_DHCPV6_TIME_SERVER_OPTION
Time zone	41	NX_DHCPV6_TIME_ZONE_OPTION

The host application can retrieve the data from the Server by calling the appropriate NetX Duo DHCPv6 Client API. An example is shown below:

```
nx_dhcpv6_get_other_option_data(
      &dhcp_0, NX_DHCPV6_DOMAIN_NAME, buffer, &address_status);
```

DHCPv6 Message Validation

All Client messages

Transaction ID: The NetX Duo DHCPv6 Client generates a transaction ID for each message to send to the Server and inserts this value into the message. DHCPv6 Server will reject any message without a transaction ID. For the server response, the client checks that the transaction ID matches the one in the

message it sent out previously. The NetX Duo DHCPv6 Client generates these values automatically based on the host MAC address and a randomization factor.

DUIDs:
All Client messages must also include a DHCPv6 unique Identifier in each message or the DHCPv6 Server will not accept its messages. A DUID is a control block containing client MAC address, hardware type, DUID type, and a time field which guarantees the DUID to be unique. If the application does not create its own unique time field, NetX Duo DHCPv6 Client will provide one.

If the Client message is intended for a specific Server, it must include that Server's DUID in the message. Thereafter, the Client will not accept any DHCPv6 messages whose Server DUID option does not match the one chosen by the Client.

Retransmissions: The NetX Duo DHCPv6 Client is configured with initial and maximum retransmission timeouts (in seconds) for waiting for server replies. It can also limit retransmissions with a maximum retransmission count. After each transmission to the server, the Client waits for the specified time. If no valid response is received, it doubles the timeout, plus a small randomizing factor, up to and until the maximum timeout is reached and/or the maximum number or retransmission attempts has been exceeded.

Below is a list of configurable options for the *Solicit* message. The complete list for all DHCPv6 messages is in the *nxd_dhcpv6_client.h*file:

```
NX_DHCPV6_INIT_SOL_TRANSMISSION_TIMEOUT
NX_DHCPV6_MAX_SOL_RETRANSMISSION_TIMEOUT
NX_DHCPV6_MAX_SOL_RETRANSMISSION_COUNT
```

To only apply maximum retransmission timeouts, set the maximum retransmission attempts to zero. To only apply maximum retransmission attempts, set the maximum retransmission timeout to zero. To remove any limitations on retransmissions, set both of these values to zero.
Rand timeouts, max retransmission timeouts and max retry attempts:

NetX Duo DHCPv6 Client Callback Functions

nx_dhcpv6_state_change_callback

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

!! Note: since this callback is called from the DHCPv6 Client thread task, the host application must NOT call any NetX Duo DHCPv6 services directly from the callback. See the Small Example for how to utilize the state change callback.

nx_dhcpv6_server_error_handler

When the DHCPv6 Client receives a Server reply containing a *Status* option with a non-zero status, it notifies the host application and forwards the server error code with it. See **Appendix B** for DHCPv6 Server error codes.

nx_dhcpv6_deprecated_IP_address_handler

When the NetX Duo DHCPv6 Client task detects that the IPv6 lease time has reached the preferred lifetime expiration time out, it notifies the host application by calling this function. At this time, the host IPv6 address is deprecated, although the host may still use it in network transmissions.

Note: since this callback is called from the DHCPv6 Client thread task, the host application should not make blocking calls from this callback, or the DHCPv6 Client is suspended on the host application and will lose responsiveness. See the Small Example for how to utilize the deprecated IP address callback.

nx_dhcpv6_expired_IP_address_handler

When the NetX Duo DHCPv6 Client task detects that the IPv6 lease time has reached the valid lifetime expiration time out, it notifies the host application by calling this function. At this time, the host IPv6 address is expired, and the host must not use it in network transmissions. Note that before calling the expiration handler, the NetX Duo DHCPv6 Client has already removed the assigned IPv6 address, and reset its address type with NetX Duo as link local.

Note: since this callback is called from the DHCPv6 Client thread task, the host application should not make blocking calls from this callback, or the DHCPv6 Client is suspended on the host application and will lose responsiveness See the Small Example for how to utilize the expired IP address callback.

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 DuoDHCPv6Client
nxd_dhcpv6_client.c	Source code file for NetX Duo DHCPv6 Client
demo_netxduo_dhcpv6.c	Sample program demonstrating the setup of the NetX Duo DHCPv6 Client and DHCPv6 server
nxd_dhcpv6_client.pdf	PDF description of NetX Duo DHCPv6 Client

NetX Duo DHCPv6 Client Installation

In order to use NetX Duo DHCPv6 Client API, the entire distribution mentioned previously should be copied to the same directory where NetX Duo is installed. For example, if NetX Duo is installed in the directory "*threadx**arm7**green*" then the *nxd_dhcpv6_client.h* and *nxd_dhpcv6_client.c* files should be copied into this directory.

Using the NetX Duo DHCPv6 Client

Using the NetX Duo DHCPv6Client API is easy. The application code must include *nxd_dhcpv6_client.h* after it includes *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX Duo, respectively. Once *nxd_dhcpv6_client.h* is included, the application code is then able to invoke DHCPv6 services. The project must also include *nxd_dhcpv6_client.c* in the build process. This file must be compiled in the same manner as other application files and its object form must be linked along with the files of the application. This is all that is required to use NetX Duo DHCPv6.

Note that since DHCPv6 utilizes NetX Duo, IPv6 and ICMPv6 must be enabled by calling *nxd_ipv6_enable* and *nxd_icmp_enable*. NetX Duo UDP services are

also utilized. Therefore, UDP must also be enabled with the *nx_udp_enable* service prior to starting the DHCPv6 Client.

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 Server running over a virtual “RAM” driver. This demo assumes a single homed host using the NetX Duo environment.

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

The DHCPv6 Client is created in line 176 that defines three callback functions for DHCP state change, address deprecation and address expiration. In the Client thread entry function, the Client IP is set up with a link local address and enabled for IPv6 and ICMPv6 services in lines 229-239.

Before starting the DHCPv6 Client, the host application creates a Client DUID in line 243 and an address block (IA-NA/IA) in line 259 - 276, all of which are necessary in DHCPv6. Then it is ready to start the DHCPv6 Client task on line 294. It sets various network parameters it would like to request from the DHCPv6 server in lines 304-307. The solicitation begins with the call to *nx_dhcpv6_request_solicit* on line 310. The DHCPv6 Client will send out the request and handle the exchanges between DHCPv6 Client and Server till either the Client receives an assigned address, or the maximum retries on soliciting the server is exceeded (see configuration options for details).

Note: The host application can optionally specify a preferred address and the IP address lease length IPv6 lease. However in this demo it is not specified (up to the Server).

In this demo, the host application polls the Client task for an assigned address in lines 323 – 338 using the *nx_dhcpv6_get_IP_address* service. It could also use the state change callback to indicate the Client is bound to an address on line 724. On being bound to an IP address, the Client must register this address with NetX Duo using the *nx_dhcpv6_register_IP_address* on line 345. If Duplicate Address Detection is enabled, NetX Duo will check if this address is unique on the Client network.

After the host application queries the DHCPv6 Client for network data on lines 363-380, it decides to release the data using the *nx_dhcpv6_request_release* service on line 395. It could alternatively determine that the address was not unique and need to decline it (line 417).

If the host application needs to stop the DHCPv6 Client for any reason, lack of server response for example, it calls the *nx_dhcpv6_stop* service on line 440.

This resets the DHCPv6 Client to the INIT (not bound) address state. The *nx_dhcpv6_reinitialize* service is called on line 450 to clear the assigned IP address from the Client IP address table. Had the host was assigned an IP address previously, it can insert it to the DHCPv6 Client IA block, on line 460. Either way, it can now restart the DHCPv6 Client by calling *nx_dhcpv6_start* and *nx_dhcpv6_request_solicit* on lines 470 and 480 respectively.

NOTE 1: the host application must NOT make DHCPv6 Client service calls from the state change callback, because this function is called from the DHCPv6 Client processing loop and will likely result in the DHCPv6 Client application locking up. In general, any blocking calls should be avoided in the state change callback. In the small example, the state change callback simply sets a flag notifying a change in Client status which the client thread can poll and then safely make API calls from the host application client entry thread.

NOTE 2: when the DHCPv6 Client address lease reaches the preferred lifetime expiration, the address deprecation callback is called, and the host application will know it needs to call *nx_dhcpv6_request_renew*. If the address lease reaches the valid lifetime expiration, for example if the renew request failed or the host elected to ignore the address deprecation notification, the address expiration callback is called and the host application will thus know to call the *nx_dhcpv6_request_rebind* service.

For details on creating and running the NetX Duo DHCPv6 Server see the *nxd_dhcpv6_server.pdf* file distributed on with the DHCPv6 Client.

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

```
30
31
32  /* Define some global flags. */
33
34  UINT g_declined = NX_FALSE;
35  UINT g_released = NX_FALSE;
36  UINT g_dhcp_failed = NX_FALSE;
37  UINT g_client_bound = NX_FALSE;
38
39  /* Define a counter for DHCP state changes. */
40  UINT   state_changes;
41  /* Define the error counter used in the demo application...  */
42  ULONG  error_counter;
43
44  /* Define thread prototypes.  */
45
46  void    thread_client_entry(ULONG thread_input);
47  void    thread_server_entry(ULONG thread_input);
48  void    dhcpv6_state_change_notify(NX_DHCPV6 *dhcp_ptr, UINT old_state, UINT new_state);
49
50  /***** Substitute your ethernet driver entry function here *********/
51  VOID    _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
52
53
54  /* Define some DHCPv6 parameters. */
55
56  #define DHCPV6_IANA_ID  0xC0DEDBAD
57  #define DHCPV6_T1       NX_DHCPV6_INFINITE_LEASE
58  #define DHCPV6_T2       NX_DHCPV6_INFINITE_LEASE
59
60
61  /* Declare NetX DHCPv6 Client callbacks. */
62
63  VOID dhcpv6_deprecated_IP_address_handler(NX_DHCPV6 *dhcpv6_ptr);
64  VOID dhcpv6_expired_IP_address_handler(NX_DHCPV6 *dhcpv6_ptr);
65
66
67  /* Define main entry point.  */
68
69  int main()
70  {
71
72      /* Enter the ThreadX kernel.  */
73      tx_kernel_enter();
74  }
75
76
77  /* Define what the initial system looks like.  */
78
79  void    tx_application_define(void *first_unused_memory)
80  {
81
82  CHAR    *pointer;
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,
91              8, 8, TX_NO_TIME_SLICE, TX_AUTO_START);
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     /* Create the Server thread.  */
102     status = tx_thread_create(&thread_server, "Server thread", thread_server_entry, 0,
103             pointer, DEMO_STACK_SIZE,
104             4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
105     /* Check for IP create errors.  */
106     if (status)
107     {
108         error_counter++;
109         return;
110     }
111
112     pointer =  pointer + DEMO_STACK_SIZE;
113
114     /* Initialize the NetX system.  */
115     nx_system_initialize();
116
117     /* Create a packet pool.  */
118     status =  nx_packet_pool_create(&pool_0, "NetX Main Packet Pool", 1024, pointer,
                                                NX_DHCPV6_PACKET_POOL_SIZE);
119     pointer = pointer + NX_DHCPV6_PACKET_POOL_SIZE;
120
121     /* Check for pool creation error.  */
122     if (status)
123         error_counter++;
124
125     /* Create a Client IP instance.  */
126     status = nx_ip_create(&client_ip, "Client IP", IP_ADDRESS(0, 0, 0, 0),
127                           0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
128                           pointer, 2048, 1);
129
130     pointer =  pointer + 2048;
131
132     /* Check for IP create errors.  */
133     if (status)
134     {
135         error_counter++;
136         return;
137     }
138
139     /* Create a Server IP instance.  */
140     status = nx_ip_create(&server_ip, "Server IP", IP_ADDRESS(1, 2, 3, 4),
141                           0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
142                           pointer, 2048, 1);
143
144     pointer =  pointer + 2048;
145
146     /* Check for IP create errors.  */
147     if (status)
148     {
149         error_counter++;
150         return;
151     }
152
153     /* Enable UDP traffic for sending DHCPv6 messages.  */
154     status =  nx_udp_enable(&client_ip);
155     status +=  nx_udp_enable(&server_ip);
156
157     /* Check for UDP enable errors.  */
158     if (status)
159     {
160         error_counter++;
161         return;
162     }
163
164     /* Enable ICMP.  */
165     status =  nx_icmp_enable(&client_ip);
166     status +=  nx_icmp_enable(&server_ip);
```

```
167
168     /* Check for ICMP enable errors.  */
169     if (status)
170     {
171         error_counter++;
172         return;
173     }
174
175     /* Create the DHCPv6 Client. */
176     status =  nx_dhcpv6_client_create(&dhcp_client, &client_ip, "DHCPv6 Client", &pool_0, pointer,
                                          NX_DHCPV6_THREAD_STACK_SIZE,
177                                       dhcpv6_state_change_notify, NX_NULL,
178                                       dhcpv6_deprecated_IP_address_handler,
179                                       dhcpv6_expired_IP_address_handler);
180
181     /* Check for errors.  */
182     if (status)
183     {
184         error_counter++;
185         return;
186     }
187
188     /* Update the stack pointer because we need it again. */
189     pointer = pointer + NX_DHCPV6_THREAD_STACK_SIZE;
190
191     /* Create the DHCPv6 Server. */
192     status =  nx_dhcpv6_server_create(&dhcp_server, &server_ip, "DHCPv6 Server", &pool_0, pointer,
                                         NX_DHCPV6_SERVER_THREAD_STACK_SIZE, NX_NULL, NX_NULL);
193
194     /* Check for errors.  */
195     if (status != NX_SUCCESS)
196     {
197         error_counter++;
198     }
199
200     /* Update the stack pointer in case we need it again. */
201     pointer = pointer + NX_DHCPV6_SERVER_THREAD_STACK_SIZE;
202
203     /* Enable the Server IP for IPv6 and ICMPv6 services. */
204     nxd_ipv6_enable(&server_ip);
205     nxd_icmp_enable(&server_ip);
206
207     /* Yield control to DHCPv6 threads and ThreadX. */
208     return;
209 }
210
211
212 /* Define the Client host application thread. */
213
214 void    thread_client_entry(ULONG thread_input)
215 {
216
217 UINT        status;
218 NXD_ADDRESS ipv6_address;
219 ULONG       T1, T2, preferred_lifetime, valid_lifetime;
220 UCHAR       buffer[200];
221 USHORT      option_code;
222
223
224     state_changes = 0;
225
226     /* Establish the link local address for the host. The RAM driver creates
227        a virtual MAC address of 0x1122334456. */
228 #ifdef MULTIHOME_NETXDUO
229     status = nxd_ipv6_address_set(&client_ip, 0, NX_NULL, 10, NULL);
230 #else
231     status = nxd_ipv6_linklocal_address_set(&client_ip, NULL);
232 #endif
233
```

```
234      /* Let NetX Duo and the network driver get initialized. Also give the server time to get set up.
*/
235      tx_thread_sleep(300);
236
237      /* Enable the Client IP for IPv6 and ICMPv6 services. */
238      nxd_ipv6_enable(&client_ip);
239      nxd_icmp_enable(&client_ip);
240
241      /* Create a Link Layer Plus Time DUID for the DHCPv6 Client. Set time ID field
242         to NULL; the DHCPv6 Client API will supply one. */
243      status = nx_dhcpv6_create_client_duid(&dhcp_client, NX_DHCPV6_DUID_TYPE_LINK_TIME,
244                                            NX_DHCPV6_HW_TYPE_IEEE_802, 0);
245
246      if (status != NX_SUCCESS)
247      {
248          error_counter++;
249          return;
250      }
251
252
253      /* Create the DHCPv6 client's Identity Association (IA-NA) now.
254
255         Note that if this host had already been assigned in IPv6 lease, it
256         would have to use the assigned T1 and T2 values in loading the DHCPv6
257         client with an IANA block.
258      */
259      status = nx_dhcpv6_create_client_iana(&dhcp_client, DHCPV6_IANA_ID, DHCPV6_T1,  DHCPV6_T2);
260
261      if (status != NX_SUCCESS)
262      {
263          error_counter++;
264          return;
265      }
266
267      memset(&ipv6_address,0x0, sizeof(NXD_ADDRESS));
268      ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
269
270      /* Create an IA address option.
271
272          Note that if this host had already been assigned in IPv6 lease, it
273          would have to use the assigned IPv5 address, preferred and valid lifetime
274          values in loading the DHCPv6 Client with an IA block.
275      */
276      status = nx_dhcpv6_create_client_ia(&dhcp_client, &ipv6_address, NX_DHCPV6_RENEW_TIME,
277                                          NX_DHCPV6_REBIND_TIME);
278
279      if (status != NX_SUCCESS)
280      {
281          error_counter++;
282          return;
283      }
284
285      /* Starting up the NetX DHCPv6 Client. */
286
287      /* If the host has no IPv6 address assigned, set the time expired to zero.
288
289         If the host has been assigned an IPv6 lease, the host needs to supply time expired on the
290         lease since stopping the Client when starting the Client back up. This may require access
291         to a real time clock or other component. */
292
293      /* Start the NetX DHCPv6 Client.  */
294      status =  nx_dhcpv6_start(&dhcp_client, 0);
295
296      /* Check for errors. */
297      if (status != NX_SUCCESS)
298      {
299
300          error_counter++;
301          return;
```

```
302     }
303
304     /* Set the list of desired options to enabled. */
305     nx_dhcpv6_request_option_timezone(&dhcp_client, NX_TRUE);
306     nx_dhcpv6_request_option_DNS_server(&dhcp_client, NX_TRUE);
307     nx_dhcpv6_request_option_time_server(&dhcp_client, NX_TRUE);
308     nx_dhcpv6_request_option_domain_name(&dhcp_client, NX_TRUE);
309
310     /* No, so the host should solicit a DHCPv6 server who will assign it one. */
311     status = nx_dhcpv6_request_solicit(&dhcp_client);
312
313     /* Check status.  */
314     if (status != NX_SUCCESS)
315     {
316
317         error_counter++;
318         return;
319     }
320
321     /* Use this service to query the DHCPv6 Client for assigned address and network data.
322        Alternatively the host application can wait for the g_client_bound signal to be set
323        by the address change notify callback (see below). */
324     do
325     {
326         /* Check if the DHCP Client is assigned an address yet. */
327          status = nx_dhcpv6_get_IP_address(&dhcp_client, &ipv6_address);
328
329          /* Check if we got a signal the DHCP client failed to get an IP address,
330             e.g. retry count exceeded without a response. */
331          if (g_dhcp_failed)
332          {
333              /* It did. We will need to restart the DHCP Client. */
334              break;
335          }
336
337          tx_thread_sleep(100);
338
339     } while (status != NX_SUCCESS);
340
341     /* Did we get an assigned address? */
342     if (status == NX_SUCCESS)
343     {
344
345         /* Yes; Register the assigned IP address with NetX Duo.  This assumes a 64 bit prefix. */
346         status = nx_dhcpv6_register_IP_address(&dhcp_client, &ipv6_address, 64);
347
348         /* Check status.  */
349         if (status != NX_SUCCESS)
350         {
351
352             error_counter++;
353             return;
354         }
355
356         /* If duplicate address detection is enabled, give it time to verify this
357            address is unique on our network. */
358         tx_thread_sleep(400);
359
360         /* Do stuff with our assigned address. */
361         tx_thread_sleep(100);
362
363         /* Get IP address lease time. */
364         status = nx_dhcpv6_get_lease_time_data(&dhcp_client, &T1, &T2, &preferred_lifetime,
                                                   &valid_lifetime);
365
366         /* Check status.  */
367         if (status != NX_SUCCESS)
368         {
369             error_counter++;
```

```
370         }
371
372         /* Get non standard option data. */
373         memset(buffer, 0, 200);
374
375         /* Set the information request option desired from Client record. */
376         option_code = NX_DHCPV6_DOMAIN_NAME_OPTION;
377
378         /* Get the information request data from the Client. The address_status field indicates
379            if the DHCPv6 Client IP address was successfully registered with the DHCP server:
380            1 = VALID; 2 = INVALID indicating the rest of the option data never came through. */
381         status = nx_dhcpv6_get_other_option_data(&dhcp_client, option_code, buffer);
382
383         if ((status != NX_SUCCESS) || (strlen((const char *)buffer)==0))
384         {
385             error_counter++;
386         }
387
388         /**** If we detect another host with the same address, we should decline the address. If we
                did not, use the address till we are done with it (leaving the network) and release it.
390          *****/
391
392 #if 1 /* Releasing! */
393
394         /* Release the address back e.g. leaving the network. Send a message to
395            the server we are releasing the assigned address.  */
396         status = nx_dhcpv6_request_release(&dhcp_client);
397
398         /* Check status.  */
399         if (status != NX_SUCCESS)
400         {
401
402             error_counter++;
403             return;
404         }
405
406         /* Wait for the signal the assigned address is released. */
407         while (!g_released)
408         {
409             tx_thread_sleep(100);
410         }
411
412         /* Reset the release flag. */
413         g_released = NX_FALSE;
414
415 #else  /* Declining! */
416
417         /* If we think the address is not unique, we must decline it. */
418         status = nx_dhcpv6_request_decline(&dhcp_client);
419
420         /* Check status.  */
421         if (status != NX_SUCCESS)
422         {
423
424             error_counter++;
425             return;
426         }
427
428         /* Wait for the signal the server knows the assigned address is declined. */
429          while (!g_declined)
430          {
431              tx_thread_sleep(100);
432          }
433
434          /* Reset the decline flag. */
435          g_declined = NX_FALSE;
436
437 #endif
438     }
```

```
439
440     /* Stopping the Client task. */
441     status = nx_dhcpv6_stop(&dhcp_client);
442     /* Check status.  */
443     if (status != NX_SUCCESS)
444     {
445
446         error_counter++;
447         return;
448     }
449
450     /* Clearing out the old session. */
451     status = nx_dhcpv6_reinitialize(&dhcp_client);
452     /* Check status.  */
453     if (status != NX_SUCCESS)
454     {
455
456         error_counter++;
457         return;
458     }
459
460     /* Now request previously assigned address. This is a hint to the server. */
461     status = nx_dhcpv6_create_client_ia(&dhcp_client, &ipv6_address, NX_DHCPV6_RENEW_TIME,
                                            NX_DHCPV6_REBIND_TIME);
462     /* Check status.  */
463     if (status != NX_SUCCESS)
464     {
465
466         error_counter++;
467         return;
468     }
469
470     /* Starting up the Client task. */
471     status = nx_dhcpv6_start(&dhcp_client, 0);
472     /* Check status.  */
473     if (status != NX_SUCCESS)
474     {
475
476         error_counter++;
477         return;
478     }
479
480     /* Soliciting an IP address with a 'hint' of what address we'd like. */
481     status = nx_dhcpv6_request_solicit(&dhcp_client);
482     /* Check status.  */
483     if (status != NX_SUCCESS)
484     {
485
486         error_counter++;
487         return;
488     }
489
490     do
491     {
492
493          status = nx_dhcpv6_get_IP_address(&dhcp_client, &ipv6_address);
494
495          tx_thread_sleep(100);
496
497     } while (status != NX_SUCCESS);
498
499
500     /* Wait a bit before releasing the IP address and terminating the client. */
501     tx_thread_sleep(100);
502
503     /* Ok, lets stop the host application. In this case we DO NOT plan
504        to keep the IPv6 address we were assigned to release it
505        back to the DHCPv6 server. */
506     status = nx_dhcpv6_request_release(&dhcp_client);
```

```
507
508     /* Check for error. */
509     if (status != NX_SUCCESS)
510     {
511         error_counter++;
512     }
513
514
515     /* Now delete the DHCPv6 client and release ThreadX and NetX resources back to
516        the system. */
517     nx_dhcpv6_client_delete(&dhcp_client);
518
519     return;
520
521 }
522
523 /* Define the test server thread. */
524 void    thread_server_entry(ULONG thread_input)
525 {
526
527 UINT        status;
528 NXD_ADDRESS ipv6_address_primary, dns_ipv6_address;
529 ULONG       duid_time;
530 UINT        addresses_added;
531 NXD_ADDRESS start_ipv6_address;
532 NXD_ADDRESS end_ipv6_address;
533
534
535     /* Wait till the IP task thread has had a chance to set the device MAC address. */
536     tx_thread_sleep(100);
537
538     /* Make the Server IP IPv6 and ICMPv6 enabled. */
539     nxd_ipv6_enable(&server_ip);
540     nxd_icmp_enable(&server_ip);
541
542     memset(&ipv6_address_primary,0x0, sizeof(NXD_ADDRESS));
543
544     ipv6_address_primary.nxd_ip_version = NX_IP_VERSION_V6 ;
545     ipv6_address_primary.nxd_ip_address.v6[0] = 0x20010db8;
546     ipv6_address_primary.nxd_ip_address.v6[1] = 0xf101;
547     ipv6_address_primary.nxd_ip_address.v6[2] = 0x00000000;
548     ipv6_address_primary.nxd_ip_address.v6[3] = 0x00000101;
549
550     /* Set the link local and global addresses. */
551
552 #ifndef NETXDUO_MULTIHOME_SUPPORT
553
554     status = nxd_ipv6_linklocal_address_set(&server_ip, NULL);
555
556     status += nxd_ipv6_global_address_set(&server_ip, &ipv6_address_primary, 64);
557
558
559 #else
560
561     status = nxd_ipv6_address_set(&server_ip, 0, NX_NULL, 10, NULL);
562
563     status += nxd_ipv6_address_set(&server_ip, 0, &ipv6_address_primary, 64, NULL);
564
565 #endif  /* NETXDUO_MULTIHOME_SUPPORT */
566
567     /* Check for errors. */
568     if (status != NX_SUCCESS)
569     {
570
571         error_counter++;
572         return;
573     }
574
575     /* Note this example assumes a single global IP address on the primary interface.  If otherwise
```

```
576         the host should call the service to set the network interface and global IP address index.
577
578          UINT _nx_dhcpv6_server_interface_set(NX_DHCPV6_SERVER *dhcpv6_server_ptr, UINT
                                                  interface_index, UINT address_index)
579      */
580
581      /* Validate the link local and global addresses. */
582      tx_thread_sleep(500);
583
584      /* Set up the DNS IPv6 server address. */
585      dns_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
586      dns_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
587      dns_ipv6_address.nxd_ip_address.v6[1] = 0x0000f101;
588      dns_ipv6_address.nxd_ip_address.v6[2] = 0x00000000;
589      dns_ipv6_address.nxd_ip_address.v6[3] = 0x00000107;
590
591      status = nx_dhcpv6_create_dns_address(&dhcp_server, &dns_ipv6_address);
592
593      /* Check for errors. */
594      if (status != NX_SUCCESS)
595      {
596
597          error_counter++;
598          return;
599      }
600
601       /* Note: For DUID types that do not require time, the 'duid_time' input can be left at zero.
602          The DUID_TYPE and HW_TYPE are configurable options that are user defined in
              nx_dhcpv6_server.h.  */
603
604      /* Set the DUID time as the start of the millenium. */
605      duid_time = SECONDS_SINCE_JAN_1_2000_MOD_32;
606      status = nx_dhcpv6_set_server_duid(&dhcp_server,
607                                         NX_DHCPV6_SERVER_DUID_TYPE, NX_DHCPV6_SERVER_HW_TYPE,
608                                         dhcp_server.nx_dhcpv6_ip_ptr -> nx_ip_arp_physical_address_msw,
609                                         dhcp_server.nx_dhcpv6_ip_ptr -> nx_ip_arp_physical_address_lsw,
610                                         duid_time);
611      if (status != NX_SUCCESS)
612      {
613          error_counter++ ;
614          return;
615      }
616
617      start_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
618      start_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
619      start_ipv6_address.nxd_ip_address.v6[1] = 0x00000f101;
620      start_ipv6_address.nxd_ip_address.v6[2] = 0x0;
621      start_ipv6_address.nxd_ip_address.v6[3] = 0x00000110;
622
623      end_ipv6_address.nxd_ip_version = NX_IP_VERSION_V6 ;
624      end_ipv6_address.nxd_ip_address.v6[0] = 0x20010db8;
625      end_ipv6_address.nxd_ip_address.v6[1] = 0x0000f101;
626      end_ipv6_address.nxd_ip_address.v6[2] = 0x00000000;
627      end_ipv6_address.nxd_ip_address.v6[3] = 0x00000120;
628
629      status = nx_dhcpv6_create_ip_address_range(&dhcp_server, &start_ipv6_address, &end_ipv6_address,
                                                    &addresses_added);
630
631      if (status != NX_SUCCESS)
632      {
633          error_counter++ ;
634          return;
635      }
636
637      /* Start the NetX DHCPv6 server!  */
638      status =  nx_dhcpv6_server_start(&dhcp_server);
639
640      /* Check for errors. */
641      if (status != NX_SUCCESS)
```

```
642     {
643         error_counter++;
644     }
645
646     /* Server is running! */
647     return;
648 }
649
650 /* Start of DHCPv6 Client callbacks. */
651
652 /*  This is an optional user defined callback the NetX DHCPv6 Client will call when it
653     detects the Client task has changed state. */
654 void dhcpv6_state_change_notify(NX_DHCPV6 *dhcpv6_ptr, UINT old_state, UINT new_state)
655 {
656
657     /* Increment state changes counter.  */
658     state_changes++;
659
660     switch (old_state)
661     {
662         case NX_DHCPV6_STATE_SENDING_REQUEST:
663         {
664             /* Check if the request failed and client state set back to INIT. */
665             if (new_state == NX_DHCPV6_STATE_INIT)
666             {
667
668                 g_dhcp_failed = NX_TRUE;
669             }
670
671             return;
672
673         }
674         case NX_DHCPV6_STATE_SENDING_RENEWAL:
675         {
676             /* Check if the request failed and client state set back to INIT. */
677             if (new_state == NX_DHCPV6_STATE_INIT)
678             {
679                 /* Address still valid; use the assigned address until the valid timeout expires. */
680             }
681
682             return;
683         }
684
685         case NX_DHCPV6_STATE_SENDING_REBIND:
686         {
687             /* Check if the request failed and client state set back to INIT. */
688             if (new_state == NX_DHCPV6_STATE_INIT)
689             {
690                 /* Indicate that the rebind failed. */
691                 g_dhcp_failed = NX_TRUE;
692             }
693
694             return;
695
696         }
697         case NX_DHCPV6_STATE_SENDING_DECLINE:
698         {
699
700             /* Client address is declined!*/
701
702             if (new_state == NX_DHCPV6_STATE_INIT)
703             {
704                 g_declined = NX_TRUE;
705             }
706
707             break;
708         }
709
710         case NX_DHCPV6_STATE_SENDING_RELEASE:
```

```
711         {
712
713             /* Client address is released */
714
715             if (new_state == NX_DHCPV6_STATE_INIT)
716             {
717                 g_released = NX_TRUE;
718             }
719
720             break;
721         }
722
723         default:
724            break;
725     }
726
727     if (new_state == NX_DHCPV6_STATE_BOUND_TO_ADDRESS)
728     {
729             /* Client is bound! Set the signal the Client has been
730                assigned an IP address. Alternatively the host application
731                can continue to poll the DHCPv6 client for a non zero IP address. */
732             g_client_bound = NX_TRUE;
733             break;
734     }
736
737     return;
738 }
739
740
741 /*  This is an optional user defined callback the NetX DHCPv6 Client will call when it
742     detects the preferred lifetime of the Client IPv6 address has expired.  The Client
743     IPv6 address is now deprecated.  */
744
745 VOID dhcpv6_deprecated_IP_address_handler(NX_DHCPV6 *dhcpv6_ptr)
746 {
747
748     /* Call the renew request service. This just sets the state and returns.
749        We will be notified if it succeeded by the state change handler. */
750     nx_dhcpv6_request_renew(dhcpv6_ptr);
751
752     return;
753 }
754
755 /*  This is an optional user defined callback the NetX DHCPv6 Client will call when it
756     detects the valid lifetime of the Client IPv6 address has expired.  The Client
757     IPv6 address is now expired.  */
758
759 VOID dhcpv6_expired_IP_address_handler(NX_DHCPV6 *dhcpv6_ptr)
760 {
761
762     /* Call the rebind request service. This just sets the state and returns.
763        We will be notified if it succeeded by the state change handler. */
764     nx_dhcpv6_request_rebind(dhcpv6_ptr);
765
766     return;
767 }
```

Figure 6. Example of the NetX Duo DHCPv6 Client

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_DISABLE_ERROR_CHECKING	This option removes the basic DHCPv6 error checking. It it typically used after the application is debugged.
NX_DHCPv6_THREAD_STACK_SIZE	Size of the DHCPv6 thread's stack. By default, the size is 2048, which represents a stack of 1024 bytes.
NX_DHCPV6_THREAD_PRIORITY	Priority of the DHCPv6 thread. By default, this value specifies that the DHCPv6 thread runs at priority 1.
NX_DHCPV6_MUTEX_WAIT	Time out option for obtaining an exclusive lock on a DHCPv6 mutex protected resource. The default value is TX_WAIT_FOREVER.
NX_DHCPV6_TICKS_PER_SECOND	Ratio of ticks to seconds and 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 60.
NX_DHCPV6_SESSION_TIMER_INTERVAL	Time interval in ticks at which the session timer updates the length of time the client has been in session communicating with

	the DHCPv6 server. By default, this value is 10.
NX_DHCPV6_RENEW_TIME	Length of time the Client requests for when it must renew the lease time of the IP address assigned to it. The default time is infinite. Note that the server will not likely grant an infinite time, but knows the client wants the longest possible renew time.
NX_DHCPV6_REBIND_TIME	Length of time the Client requests for when it must rebind lease time of the IP address assigned to it. The default time is infinite. Note that the server will not likely grant an infinite time, but knows the client wants the longest possible rebind time
NX_DHCPV6_NUM_DNS_SERVERS	Number of DNS servers to store to the client record. By default, this value is1.
NX_DHCPV6_NUM_TIME_SERVERS	Number of time servers to store to the client record. By default, this 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_MESSGE_SIZE	Size of the buffer in the Client record to hold server text messages. 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_PACKET_SIZE	Size of the packet in the client

	packet pool. The default value is 500 bytes.
NX_DHCPV6_PACKET_POOL_SIZE	Size of the packet in the Client packet pool. The default value is 10 packets, so 500 bytes * 10 packets).
NX_DHCPV6_TYPE_OF_SERVICE	This defines the type of service for UDP packet transmission. By default, this value is **NX_IP_NORMAL**.
NX_DHCPV6_FRAGMENT_OPTION	Fragment enable for DHCPv6 UDP requests. By default, this value is set to **NX_DONT_FRAGMENT**.
NX_DHCPV6_TIME_TO_LIVE	Specifies the number of routers Client packets can pass before packets are discarded. The default valueis set to 0x80.
NX_DHCPV6_QUEUE_DEPTH	Specifies the number of packets to keep in the Client UDP socket queue before NetX Duo discards packets.

Each message type in the NetX Duo DHCPv6 client has an initial and maximum timeout (seconds) to wait for a valid server reply before retransmitting, as well as a maximum number of retries attempting to contact the server. Whichever limit is reached first, maximum timeout or maximum retires,will cause the client to abort the request. To remove any limit on the timeout value or retries, set the respective values to zero.

Below is listed the default values for each message, and all are configurable in *nxd_dhcpv6_client.h.*

SOLICIT	Time out (sec)
NX_DHCPV6_INIT_SOL_TRANSMISSION_TIMEOUT	1
NX_DHCPV6_MAX_SOL_RETRANSMISSION_TIMEOUT	120
NX_DHCPV6_MAX_SOL_RETRANSMISSION_COUNT	10

REQUEST	Time out (sec)
NX_DHCPV6_INIT_REQ_TRANSMISSION_TIMEOUT	1
NX_DHCPV6_MAX_REQ_RETRANSMISSION_TIMEOUT	30
NX_DHCPV6_MAX_REQ_RETRANSMISSION_COUNT	10

RENEW	Time out (sec)
NX_DHCPV6_INIT_RENEW_TRANSMISSION_TIMEOUT	2
NX_DHCPV6_MAX_RENEW_RETRANSMISSION_TIMEOUT	600
NX_DHCPV6_MAX_RENEW_RETRANSMISSION_COUNT	0

REBIND	Time out (sec)
NX_DHCPV6_INIT_REBIND_TRANSMISSION_TIMEOUT	2
NX_DHCPV6_MAX_REBIND_RETRANSMISSION_TIMEOUT	600
NX_DHCPV6_MAX_REBIND_RETRANSMISSION_COUNT	0

RELEASE	Time out (sec)
NX_DHCPV6_INIT_RELEASE_TRANSMISSION_TIMEOUT	1
NX_DHCPV6_MAX_RELEASE_RETRANSMISSION_TIMEOUT	0
NX_DHCPV6_MAX_RELEASE_RETRANSMISSION_COUNT	5

CONFIRM	Time out (sec)
NX_DHCPV6_INIT_CONFIRM_TRANSMISSION_TIMEOUT	1
NX_DHCPV6_MAX_ CONFIRM _RETRANSMISSION_TIMEOUT	4
NX_DHCPV6_MAX_ CONFIRM _RETRANSMISSION_COUNT	0

DECLINE	Time out (sec)
NX_DHCPV6_INIT_DECLINE_TRANSMISSION_TIMEOUT	1
NX_DHCPV6_MAX_ DECLINE _RETRANSMISSION_TIMEOUT	0
NX_DHCPV6_MAX_ DECLINE _RETRANSMISSION_COUNT	5

INFORM REQUEST	Time out (sec)
NX_DHCPV6_INIT_INFORM_TRANSMISSION_TIMEOUT	1
NX_DHCPV6_MAX_ INFORM _RETRANSMISSION_TIMEOUT	120
NX_DHCPV6_MAX_ INFORM _RETRANSMISSION_COUNT	0

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 DHCPv6Client instance

nx_dhcpv6_client_delete
Delete a DHCPv6Client instance

nx_dhcpv6_client_create_duid
Create a DHCPv6 Client DUID

nx_dhcpv6_client_ia
Delete a DHCPv6 Client Identity Association (IA)

nx_dhcpv6_client_iana
Delete a DHCPv6 Client Identity Association for Non Temporary Addresses (IANA)

nx_dhcpv6_get_client_duid_time_id
Get 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 IP address assigned to the DHCPv6 client

nx_dhcpv6_get_lease_time_data
Get T1, T2, valid and preferred lifetimes associated with Client’s current assigned IP address assigned to the DHCPv6 Client

nx_dhcpv6_get_other_option_data
Get specific option information from DHCPv6 Server e.g. domain name or time zone

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 on the current IP address lease since it was assigned to the DHCPv6 Client

nx_dhcpv6_reinitialize
Remove the assigned IP address from IP table

nx_dhcpv6_request_confirm
Process and send a CONFIRM request to the Server

nx_dhcpv6_request_decline
Process and send a DECLINE request to the Server

nx_dhcpv6_request_inform_request
Process and send an INFORM REQUEST message to the Server

nx_dhcpv6_request_option_DNS_server
Add the DNS server option to the Client inform request message to the Server

nx_dhcpv6_request_option_domain_name
Add the domain name option to the Client inform request message to the Server

nx_dhcpv6_request_option_time_server
Add the time server option to the Client inform request message to the Server

nx_dhcpv6_request_option_timezone
Add the time zone option to the Client inform request message to the Server

nx_dhcpv6_request_rebind
Process and send a REBIND request to the Server

nx_dhcpv6_request_renew
Process and send a RENEW request to the Server

nx_dhcpv6_request_solicit
Process and send a SOLICIT request to the Server

nx_dhcpv6_resume

Resume DHCPv6 Client processing

nx_dhcpv6_start
Start DHCPv6 Client processing

nx_dhcpv6_stop
Stop DHCPv6 Client processing

nx_dhcpv6_suspend
Suspend DHCPv6 Client processing

Nx_dhcpv6_register_IP_address
Register assigned address with IP instance

nx_dhcpv6_set_time_accrued
Set the time accrued on the current Client IP 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),
      VOID (*dhcpv6_deprecated_IP_address_handler)(struct NX_DHCPV6_STRUCT
            *dhcpv6_ptr),
      VOID (*dhcpv6_expired_IP_address_handler)(struct NX_DHCPV6_STRUCT
            *dhcpv6_ptr))
```

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
dhcpv6_deprecated_IP_address_handler
Pointer to callback function invoked when the Client IP address has become deprecated
dhcpv6_expired_IP_address_handler
Pointer to callback function invoked when the Client IP address has become expired

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,
        dhcpv6_deprecated_IP_address_handler, dhcpv6_expired_IP_address_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 by filling in the Client DUID in the Client instance with the supplied parameters.

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

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

Create Client Identity Association object

Prototype

```
UINT    _nx_dhcpv6_create_client_ia(NX_DHCPV6 *dhcpv6_ptr,
                        NXD_ADDRESS *ipv6_address,
                        ULONG preferred_lifetime,
                        ULONG valid_lifetime)
```

Description

This service creates the 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.

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 ClientIA created
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

```
/* Create the Client IA from 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 created.  */
```

See Also

nx_dhcpv6_create_client_duid, nx_dhcpv6_create_server_duid, nx_dhcpv6_create_client_iana

nx_dhcpv6_create_client_iana

Create Client Identity Association (Non Temporary) object

Prototype

```
UINT    _nx_dhcpv6_create_client_iana(NX_DHCPV6 *dhcpv6_ptr,
                      UINT IA_ident, ULONG T1, ULONG T2)
```

Description

This service creates the Client Non Temporary Identity Association (IANA) object by filling in the Client record with the supplied parameters. To request the maximum T1 and T2 times, set these parameters to infinity.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
IA_ident	Identity Association unique identifier
T1	Time at which Client must renew IP address
T2	Time at which Client must rebind IP address

Return Values

NX_SUCCESS	(0x00)	Successful Client IANA created
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_create_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 NetX Duo DHCPv6 Client DUID. If the host 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 host 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

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 as specified by the input interface index for communicating with the DHCPv6 Server. Note: this is only available for Client hosts with NetX Duo 5.6 or later.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
interface_index	Index indicating network interface

Return Values

NX_SUCCESS	(0x00)	Interface successfully set
NX_NOT_ENABLED	(0x14)	Multihoming not enabled
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 assigned global IP address

Prototype

```
UINT    _nx_dhcpv6_get_IP_address(NX_DHCPV6 *dhcpv6_ptr,
                                  NXD_ADDRESS *ip_address)
```

Description

This service retrieves the Client's assigned global IP address. If the DHCPv6 server assigned a valid address, the return status will be NX_SUCCESS. If not, an error status is returned.

Note that the host must still register the assigned IP address with NetX Duo using the *nx_dhcpv6_register_IP_address* service.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
ip_address	Pointer to NetX Duo IP address block

Return Values

NX_SUCCESS	(0x00)	IP address successfully assigned
NX_DHCPV6_CLIENT_ADDRESS_INVALID	(0xFFFFFFF)	No IP address assigned
NX_PTR_ERROR	(0x16)	Invalid pointer input

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 IP 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 assigned global IP address. The Client's address status is returned to verify the Client IP address is still valid and registered (if set to (NX_SUCCESS) with the DHCPv6 server.

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	IP lease data successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

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

/* If status is NX_SUCCESS the client IP 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_DNS_server_address

Retrieves DNS Server IP address from the DHCPv6 Client list

Prototype

```
UINT  _nx_dhcpv6_get_DNS_server(NX_DHCPV6 *dhcpv6_ptr,
        UINT index, NXD_ADDRESS *server_address)
```

Description

This service retrieves the DNS server IP address data at the specified index of the DHCPv6 Client list. The list is compiled by the Client from DHCPv6 server messages with DNS server data. If the list does not contain a server address at the index, a null IPv6 address is returned. The length of the Client 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 Client DNS server list
server_address	Pointer to copy the server address into

Return Values

NX_SUCCESS	(0x00)	Address successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

```
/* Retrieve the DNS server IP address(es) in the Client's DNS server list. */
UINT index = 0;
NXD_ADDRESS server_address[NX_DHCPV6_NUM_DNS_SERVERS - 1];

 while(index < NX_DHCPV6_NUM_DNS_SERVERS)
 {

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

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

     index++;
 }
```

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 non IP address data from DHCPv6 Client

Prototype

```
UINT  _nx_dhcpv6_get_other_option_data(NX_DHCPV6 *dhcpv6_ptr,
          UINT option_code, UCHAR *buffer)
```

Description

This service retrieves non IP address data specified by the option code the Client received from the server required for setting up its network environment, such as DNS server, time server, and domain name. The Client's address status is returned to verify the Client IP address is still valid and registered (if set to (NX_SUCCESS) with the DHCPv6 server.

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_PTR_ERROR	(0x16)	Invalid pointer input
NX_DHCPV6_PARAM_ERROR	(0xE93)	Invalid non pointer input

Allowed From

Threads

Example

```
/* Retrieve the non IP address 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 global IP address since it was assigned by the server. The Client's address status is returned to verify the Client IP address is still valid and registered (if set to (NX_SUCCESS) with the DHCPv6 server.

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	IP lease data successfully retrieved
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Retrieve time accrued since client's assigned IP address was assigned.  */
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 clears the DHCPv6 assigned IP address from the IP address from the IP address table. If a host application stops the DHCPv6 Client, and will be requesting another IP address from the DHCPv6 server, it should reinitialize the Client, before starting the DHCPv6 Client again. See the “Small Example” section for details.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Address successfully removed
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Remove the DHCP client' assigned IP address from the IP address table.  */
status = nx_dhcpv6_reinitialize(&dhcp_0);

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

See Also

nx_dhcpv6_reinitialize, 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 creates and transmits a CONFIRM message for the Client, depending on the client transmission parameters waits to receive the server reply. 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)	CONFIRM message successfully created and processed
NX_PTR_ERROR	(0x16)	Invalid pointer input

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 and processed
    the reply.  */
```

See Also

nx_dhcpv6_request_decline, nx_dhcpv6_request_inform_request, nx_dhcpv6_request_rebind, nx_dhcpv6_request_release, nx_dhcpv6_request_renew, nx_dhcpv6_request_solicit

nx_dhcpv6_request_decline

Process the Client's DECLINE message

Prototype

```
UINT _nx_dhcpv6_request_decline(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service creates and transmits a DECLINE message for the Client, depending on the Client transmission parameters waits to receive the server reply. When one is received, the reply is processed to determine it is valid and the server granted the request. The Client isntance is then updated with the server information as needed.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	DECLINE message successfully created and processed
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Send a DECLINE message to the server.  */
status = nx_dhcpv6_request_decline(&dhcp_0);

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

See Also

nx_dhcpv6_request_confirm, nx_dhcpv6_request_inform_request, nx_dhcpv6_request_rebind, nx_dhcpv6_request_release, nx_dhcpv6_request_renew, 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 creates and transmits an INFORM REQUEST message for the Client, depending on the client transmission parameters waits to receive the server reply. 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

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_decline, nx_dhcpv6_request_rebind, nx_dhcpv6_request_release,
nx_dhcpv6_request_renew, nx_dhcpv6_request_solicit

nx_dhcpv6_request_option_DNS_server

Set DNS server(s) data as optional request

Prototype

```
UINT _nx_dhcpv6_request_option_DNS_server(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service creates and transmits either a SOLICIT, RENEW, REBIND or INFORM REQUEST message for the Client, containing the option for requesting DNS server information. Depending on the client transmission parameters, the Client waits to receive the server reply. 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 DNS server information as needed. How many DNS server IP addresses can be stored in the Client instance is a configurable option (NX_DHCPV6_NUM_DNS_SERVERS).

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client message successfully created with DNS server request included
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Set DNS server data as one of the optional requests from the DHCPv6 server.  */
_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

Set domain name data as optional request

Prototype

```
UINT _nx_dhcpv6_request_option_domain_name(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service creates and transmits either a SOLICIT, RENEW, REBIND or INFORM REQUEST message for the Client, containing the option for requesting domain name information. Depending on the Client transmission parameters, the Client waits to receive the server reply. 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 domain name information as needed. The size of the buffer for holding the domain name information is a configurable option (NX_DHCPV6_DOMAIN_NAME_BUFFER_SIZE).

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client message successfully created with domain name request included
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Set domain name data as one of the optional requests from the DHCPv6 server.  */
_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 creates and transmits either a SOLICIT, RENEW, REBIND or INFORM REQUEST message for the Client, containing the option for requesting time server information. Depending on the Client transmission parameters, the Client waits to receive the server reply. 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 time server information as needed. The number of time servers that can be stored in the Client instance is a configurable option (NX_DHCPV6_NUM_TIME_SERVERS).

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client message successfully created with time server request included
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Set time server data as one of the optional requests from the DHCPv6 server.  */
_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 creates and transmits either a SOLICIT, RENEW, REBIND or INFORM REQUEST message for the Client, containing the option for requesting time zone information. Depending on the Client transmission parameters, the Client waits to receive the server reply. 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 time zone information as needed. The size of the buffer to hold time zone data is a configurable option (NX_DHCPV6_TIME_ZONE_BUFFER_SIZE).

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client message successfully created with time zone request included
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Set time zone data as one of the optional requests from the DHCPv6 server.  */
_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_rebind

Process the Client's REBIND state

Prototype

```
UINT _nx_dhcpv6_request_rebind(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service creates and transmits a REBIND message for the Client, depending on the client transmission parameters waits to receive the any server's reply. 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)	REBIND message successfully created and processed
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Send an REBINDmessage to the server.  */
status = nx_dhcpv6_request_rebind(&dhcp_0);

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

See Also

nx_dhcpv6_request_confirm, nx_dhcpv6_request_decline, nx_dhcpv6_request_inform_request, nx_dhcpv6_request_release, nx_dhcpv6_request_renew, nx_dhcpv6_request_solicit

nx_dhcpv6_request_release

Process the Client's RELEASE state

Prototype

```
UINT _nx_dhcpv6_request_renew(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service creates and transmits a RELEASE message for the Client, depending on the Client transmission parameters waits to receive the server reply. The Client is not required to wait for a server reply. However if it does, 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)	RELEASE message successfully created and processed
NX_PTR_ERROR	(0x16)	Invalid pointer input

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 and processed
    the reply.  */
```

See Also

nx_dhcpv6_request_confirm, nx_dhcpv6_request_decline, nx_dhcpv6_request_inform_request, nx_dhcpv6_request_renew, nx_dhcpv6_request_rebind, nx_dhcpv6_request_solicit

nx_dhcpv6_request_renew

Process the Client's RENEW state

Prototype

```
UINT _nx_dhcpv6_request_renew(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service creates and transmits a RENEW message for the Client, depending on the Client transmission parameters waits to receive the server reply. 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)	RENEW message successfully created and processed
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Send an RENEW message to the server.  */
status = nx_dhcpv6_request_renew(&dhcp_0);

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

See Also

nx_dhcpv6_request_confirm, nx_dhcpv6_request_decline, nx_dhcpv6_request_inform_request, nx_dhcpv6_request_release, nx_dhcpv6_request_rebind, nx_dhcpv6_request_solicit

nx_dhcpv6_request_solicit

Process the Client's SOLICIT state

Prototype

```
UINT _nx_dhcpv6_request_solicit(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service creates and transmits a SOLICIT message for the Client, depending on the Client transmission parameters waits to receive a server's ADVERTISE reply. This is generally how the Client requests a global IP address be assigned to it.

When one is received that the Client considers valid, the Client also then sends a REQUEST message and waits for a reply from the same server. That 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)	SOLICIT message successfully created and processed
NX_PTR_ERROR	(0x16)	Invalid pointer input

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 and processed
    the reply.  */
```

See Also

nx_dhcpv6_request_confirm, nx_dhcpv6_request_decline, nx_dhcpv6_request_inform_request, nx_dhcpv6_request_release, nx_dhcpv6_request_rebind, nx_dhcpv6_request_new

nx_dhcpv6_resume

Resume DHCPv6 Client task

Prototype

```
UINT _nx_dhcpv6_resume(NX_DHCPV6 *dhcpv6_ptr)
```

Description

This service resumes the DHCPv6 client task and any request that the client was in the middle of processing.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client successfully resumed
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

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

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

See Also

nx_dhcpv6_start, nx_dhcpv6_suspend

nx_dhcpv6_register_IP_address

Registers the Client's assigned IP address with IP task

Prototype

```
UINT _nx_dhcpv6_register_IP_address(NX_DHCPV6 *dhcpv6_ptr,
                        ULONG prefix)
```

Description

This service registers the DHCPv6 Client's address with the IP instance (inserts the global address into the IP address table). This is required for the DHCPv6 Client to be able to use the IP address. Note: the host application is responsible for verifying the assigned IP address is unique on the host network. If not, it can use the nx_dhcpv6_request_decline to reject the address. See the "Small Example" section for more details.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance
prefix_length	Length of IPv6 address prefix, usually 64

Return Values

NX_SUCCESS	(0x00)	Address successfully registered
NX_IP_ADDRESS_ERROR	(0x21)	Invalid address input
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Register the IP address with the IP instance.  */
status = nx_dhcpv6_register_IP_address(&dhcpv6_0, 64);

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

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_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 would be used when a Client already configured with an assigned valid IP address is powered back on and needs to update the time accrued on its IP lease with the DHCPv6 client task.

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 readies the client to process application requests for sending DHCPv6 messages. It verifies the Client instance has sufficient information (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 IP lease expires.

Input Parameters

dhcpv6_ptr	Pointer to DHCPv6 Client instance

Return Values

NX_SUCCESS	(0x00)	Client successfully started
NX_PTR_ERROR	(0x16)	Invalid pointer input

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, clears any session parameters such as retry count, 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_PTR_ERROR	(0x16)	Invalid pointer input

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

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 – DHCPv6 Option Codes

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

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

Appendix B - DHCPv6 Server Status Codes

Name	Code	Description
Success	0	Success
Unspecified Failure	1	Failure, reason unspecified; this statuscode is sent by either a client or a server to indicate a failure not explicitly specified in this document
NoAddress Available	2	Server has no addresses available to assign to the IA(s)
NoBinding	3	Client record (binding) unavailable
NotOnLink	4	The prefix for the address is not appropriate for the link to which the client is attached
UseMulticast	5	Sent by a server to a client to force the client to send messages to the server using the All_DHCP_Relay_Agents_and_Servers address

Appendix C - DHCPv6 Unique Identifiers (DUIDs)

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

Enterprise DUID types are not supported in this release of NetX Duo DHCPv6 Client.

Appendix D – NetX Duo DHCPv6 Client States

NX_DHCPV6_STATE_INIT	1
NX_DHCPV6_STATE_SENDING_SOLICIT	2
NX_DHCPV6_STATE_SENDING_REQUEST	3
NX_DHCPV6_STATE_SENDING_RENEWAL	4
NX_DHCPV6_STATE_SENDING_REBIND	5
NX_DHCPV6_STATE_SENDING_DECLINE	6
NX_DHCPV6_STATE_SENDING_CONFIRM	7
NX_DHCPV6_STATE_SENDING_INFORM_REQUEST	8
NX_DHCPV6_STATE_SENDING_RELEASE	9
NX_DHCPV6_STATE_SUCCESS	12
NX_DHCPV6_STATE_IP_FAILURE	13
NX_DHCPV6_STATE_OTHER_FAILURE	14