Renesas Synergy™ Platform

# NetX Duo™ DHCP Client Module Guide

R11AN0142EU0100
Rev.1.00
Jan 9, 2018

## Introduction

This Module Guide will enable you to effectively use a module in your own design. Upon completion of this guide, you will be able to add this module to your own design, configure it correctly for the target application, and write code using the included application project code as a reference and an efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advanced uses of the module are included in this document and should be valuable resources for creating more complex designs.

### DHCP IPv4 Client

The Dynamic Host Configuration Protocol (DHCP) is a useful protocol to obtain IP addresses and network parameters. DHCP extends the basic functionality of BOOTP from a static address configuration to a dynamic IP address allocation by "leasing" an IP address to a client for a specified period of time. The DHCP can also be configured to allocate IP addresses in a static manner like BOOTP. An application's IP address is one of the supplied parameters for the NetX Duo™ component; supplying an IP address poses no problem when the IP address is known to the application, either statically or through user-configuration. When the application doesn't know or care what its IP address is the NetX Duo is initialized with a zero IP address; a DHCP client component added to NetX Duo can then dynamically obtain an IP address.

Note: The NetX™ DHCP Client is almost identical to the NetX Duo DHCP Client. NetX DHCP Client applications should build and run without any changes in a NetX Duo environment.

This Renesas Synergy Platform module guide covers key elements related to the NetX Duo DHCP IPv4 Client implementation, with the primary focus on the addition and configuration of the NetX Duo DHCP Client IPv4 Module to a project. For details on module operation, review, "*NetX Duo Dynamic Host Configuration Protocol for Clients User Guide for the Renesas Synergy™ Platform*." This document is available from Renesas Synergy Gallery (https://synergygallery.renesas.com/ssp/support#read) as part of the X-Ware™ and NetX Duo™ Component Documents for Renesas Synergy™ zip file.

### DHCP IPv6 Client

The document covers the NetX Duo DHCPv6 Client API and how it is used to obtain IPv6 addresses. In IPv6 networks, DHCPv6 (instead of DHCP) is used for dynamic global IPv6 address assignment from a DHCPv6 server. DHCPv6 offers many of the same features, as well as several enhancements.

This Renesas Synergy Platform module guide covers key elements related to the NetX Duo DHCPv6 Client implementation, with its primary focus on the addition and configuration of the NetX DUO DHCPv6 Client module to a project. For details on module operation, review the "*NetX Duo™ Dynamic Host Protocol (DHCPv6) Client User Guide for the Renesas Synergy™ Platform*." This document is available from the Renesas Synergy Gallery (https://synergygallery.renesas.com/ssp/support#read) as part of the X-Ware™ and NetX™ Component Documents for Renesas Synergy™ zip file.

## Contents

## 1. NetX Duo DHCP Client Module Features

### DHCP IPv4 Client

- The NetX Duo DHCP IPv4 Client module is compliant with RFC2132, RFC2131, and related RFCs.
- Support for IPv4
- Provides high-level APIs to:
  - Create and delete a DHCP Client instance
  - Start, stop, and reinitialize the DHCP Client (to restart the DHCP Client protocol)
  - Request a specific IP address from the server
  - Specify the network interface to run the DHCP Client on
  - Supply an application-created packet pool to the DHCP Client

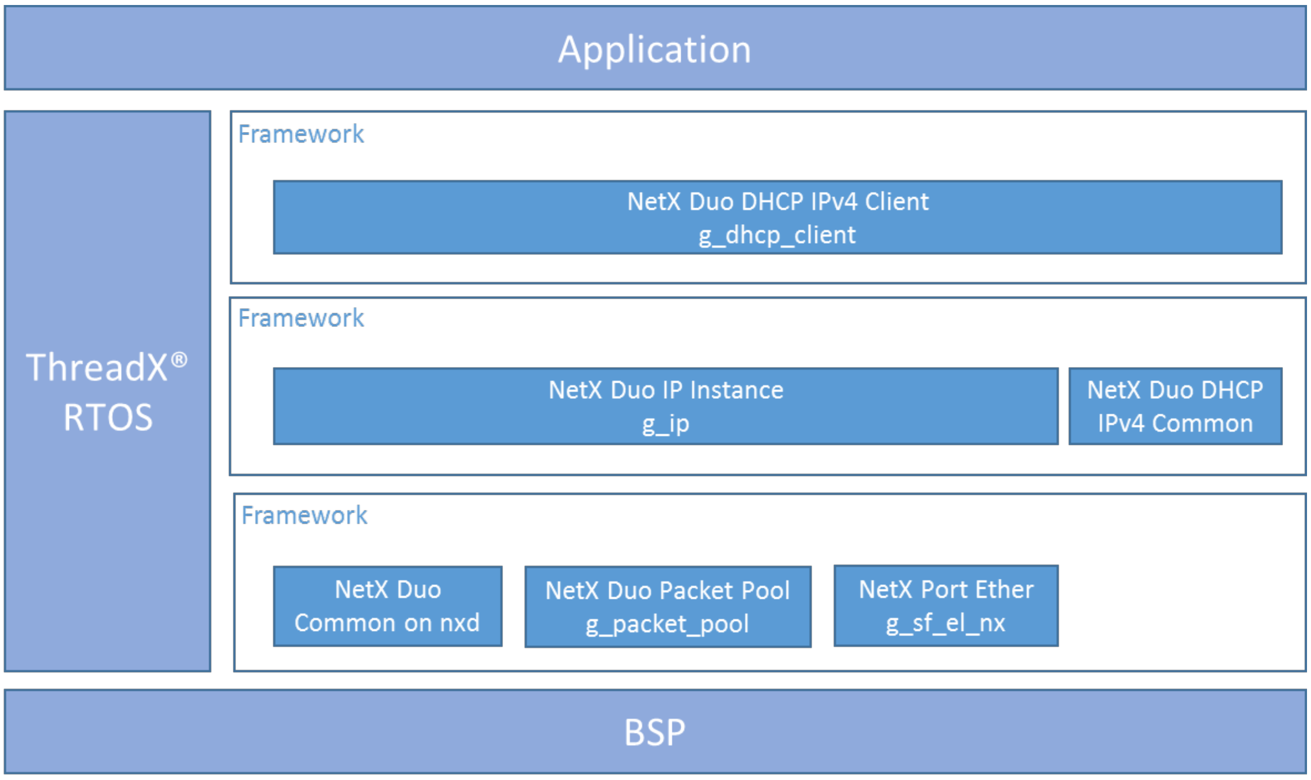

**Figure 1 NetX Duo DHCP IPv4 Client Module**

### DHCP IPv6 Client

- NetX Duo DHCPv6 Client is compliant with RFC 3315, RFC 3646, and related RFCs.
- Provides high-level APIs for:
  - Creating and deleting a DHCPv6 Client instance
  - Starting and stopping a DHCPv6 Client
  - Message sending and processing
  - Retrieving DHCPv6 data from the DHCPv6 Client

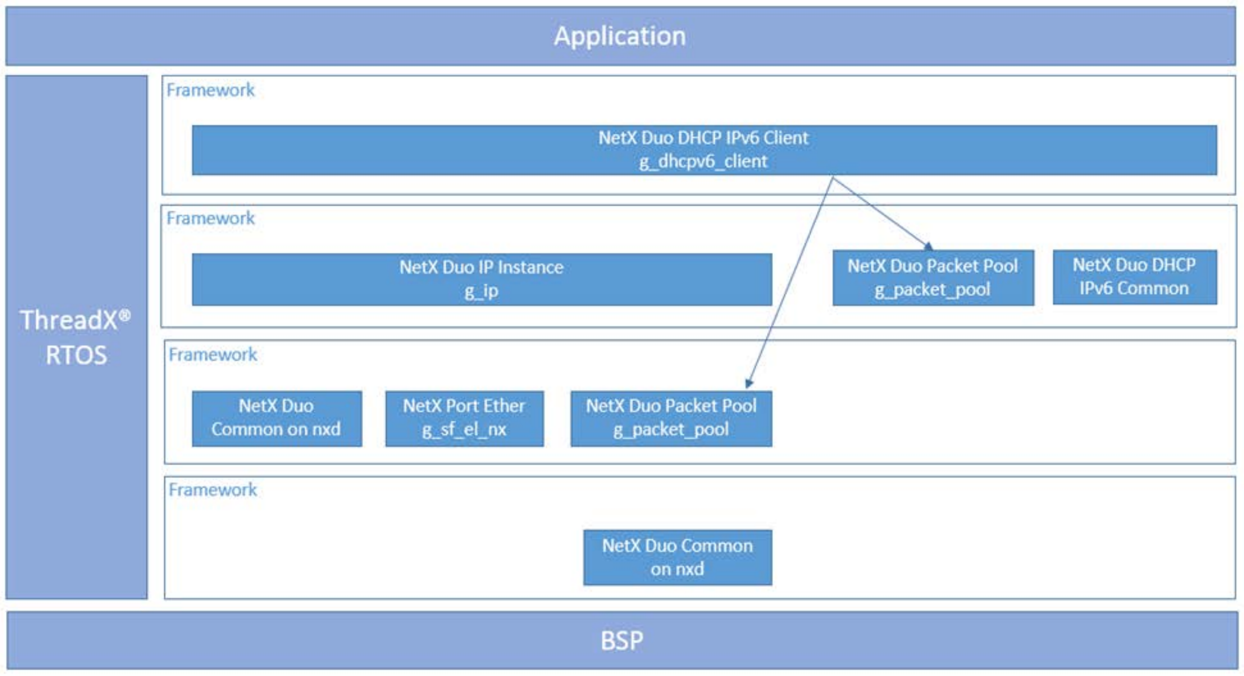

**Figure 2 NetX Duo DHCP IPv6 Client Module**

## 2. NetX Duo DHCP Client Module APIs Overview

### DHCP IPv4 Client

The NetX Duo DHCP IPv4 Client module defines APIs for creating and starting the DHCP Client. Internally, the DHCP Client handles all communication with the DHCP Server to obtain an IP address. The following table includes a complete list of the available APIs, example API calls, and a brief description of each API. A table of status return values follows.

**Table 1 NetX Duo DHCP IPv4 Client Module API Summary**

| Function Name | Example API Call and Description |
|---|---|
| nx_dhcp_create | `nx_dhcp_create(&my_dhcp, &my_ip, "My DHCP");`<br>Create a DHCP instance. |
| nx_dhcp_clear_broadcast_flag | `nx_dhcp_clear_broadcast_flag(&my_dhcp, NX_TRUE);`<br>Clear broadcast flag on Client messages. |
| nx_dhcp_delete | `nx_dhcp_delete(&my_dhcp);`<br>Delete a DHCP instance. |
| nx_dhcp_decline | `nx_dhcp_decline(&my_dhcp);`<br>Send Decline message to server. |
| nx_dhcp_force_renew | `nx_dhcp_force_renew(&my_dhcp);`<br>Handle Server force renew message. |
| nx_dhcp_packet_pool_set | `nx_packet_pool_create(&dhcp_pool, "DHCP Client Packet Pool",`<br>`NX_DHCP_PACKET_PAYLOAD, pointer, (15 * NX_DHCP_PACKET_PAYLOAD));`<br>`nx_dhcp_create(&dhcp_0, &ip_0, "janetsdhcp1");`<br>`nx_dhcp_packet_pool_set(&my_dhcp, packet_pool_ptr);`<br>Set the DHCP Client packet pool. By default, the DHCP Client creates its own packet pool. |
| nx_dhcp_release | `nx_dhcp_release(&my_dhcp);`<br>Send Release message to server. |

| Function Name | Example API Call and Description |
|---|---|
| nx_dhcp_reinitialize | `nx_dhcp_reinitialize(&my_dhcp);`<br>Clear DHCP client network parameters and clear IP address and gateway registered with the IP instance. |
| nx_dhcp_request_client_ip | `nx_dhcp_request_client_ip(&my_dhcp, IP(192,168,0,6), NX_TRUE);`<br>Request a specific IP address. |
| nx_dhcp_send_request | `nx_dhcp_send_request(&my_dhcp, NX_DHCP_TYPE_INFORMREQUEST);`<br>Send DHCP message to server (only INFORM_REQUEST is allowed). |
| nx_dhcp_server_address_get | `nx_dhcp_server_address_get(&dhcp_0, &server_address);`<br>Retrieve DHCP Client's DHCP server address. |
| nx_dhcp_set_interface_index | `nx_dhcp_set_interface_index(&my_dhcp, 1);`<br>Specify the network interface to run DHCP Client. |
| nx_dhcp_start | `nx_dhcp_start(&my_dhcp);`<br>Start DHCP processing. |
| nx_dhcp_state_change_notify | `nx_dhcp_state_change_notify(&my_dhcp, my_state_change);`<br>Notify application of DHCP state change. |
| nx_dhcp_stop | `nx_dhcp_stop(&my_dhcp);`<br>Stop DHCP processing. |
| nx_dhcp_user_option_retrieve | `nx_dhcp_user_option_retrieve(&my_dhcp, NX_DHCP_OPTION_DNS_SVR, dns_ip_string, &size);`<br>Retrieve the specified DHCP option. |
| nx_dhcp_user_option_convert | `nx_dhcp_user_option_convert(dns_ip_string);`<br>Convert four bytes to ULONG. |
| The following services require that *Persistent client state* be enabled | |
| nx_dhcp_suspend | `nx_dhcp_suspend(&g_dhcp_client0);`<br>`Suspend the DHCP Client thread.` |
| nx_dhcp_resume | `nx_dhcp_resume (&g_dhcp_client0);`<br>Resume the DHCP Client thread. |
| nx_dhcp_client_update_time_remaining | `nx_dhcp_client_update_time_remaining(*g_dhcp_client0, 1000)`<br>This updates the time remaining on the IP lease by the input time in timer ticks e.g. the time interval while the DHCP Client thread was suspended. |
| nx_dhcp_client_create_record | `nx_dhcp_client_create_record(&g_dhcp_client0)`<br>This fills in a client record structure associated with the DHCP Client based on Client lease data. |
| nx_dhcp_client_restore_record | `nx_dhcp_client_restore_record(&g_dhcp_client, client_record_ptr, ime_elapsed)`<br>The Client record points to data to restore to the DHCP Client itself, and time elapsed is subtracted from the DHCP Client time remaining on its lease. |

Note: For details on operation and definitions for function data structures, typedefs, defines, API data, API structures, and function variables, review the associated Express Logic User's Manual in the References section.

**Table 2 Status Return Values**

| Name | Description |
|---|---|
| NX_SUCCESS | Successful API call. |

| Name | Description |
|---|---|
| NX_PTR_ERROR* | Invalid pointer input. |
| NX_THREADS_ONLY_CALLER_CHECKING* | Invalid caller of this service. |
| NX_INVALID_INTERFACE | NetX is not enabled on the input interface |
| NX_NOT_ENABLED | Not enabled to set the DHCP Client packet pool. |
| NX_DHCP_NOT_STARTED | DHCP Client not started. |
| NX_DHCP_NOT_BOUND | The IP address has not been leased so the current operation is not allowed. |
| NX_DHCP_INVALID_MESSAGE | Illegal message type to send. |
| NX_DHCP_BAD_INTERFACE_INDEX* | An invalid network interface supplied |
| NX_DHCP_UNKNOWN_OPTION | Unknown DHCP option to extract from DHCP server response |
| NX_DHCP_INVALID_IP_REQUEST* | Invalid address for the DHCP Client to request |
| NX_DHCP_INVALID_PAYLOAD | Packet pool for the DHCP Client has insufficient payload |
| NX_DHCP_ALREADY_STARTED | DHCP Client thread task has already started |
| NX_DHCP_PARSE_ERROR | Unable to parse requested option from Server response |
| NX_DHCP_DEST_TO_SMALL | Supplied buffer too small to hold the requested option data for user requesting option data |

Note: Lower-level drivers may return common error codes. See the *SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

*These error codes are only returned if error checking is enabled. See the NetX Duo User Guide for the Renesas Synergy™ Platform for details on error-checking services in NetX Duo.

### DHCP IPVv6 Client

The NetX Duo DHCPv6 Client framework defines API's for creating, deleting, adding, and getting client information. The following table has a complete list of the available APIs, an example API call, and a brief description of each. A table of return status values follows.

**Table 3 NetX Duo DHCP IPv6 Client Module API Summary**

| Function Name | Example API Call and Description |
|---|---|
| nx_dhcpv6_client_create | `nx_dhcpv6_client_create(&dhcp_0, &ip_0, "DHCPv6 Client", &pool_0,NULL, NULL, pointer, 2048, dhcpv6_state_change_notify, dhcpv6_server_error_handler);` Create a DHCPv6 Client instance. |
| nx_dhcpv6_client_delete | `nx_dhcpv6_client_delete(&my_dhcp);` Delete a DHCPv6 Client instance. |
| nx_dhcpv6_client_set_interface | `nx_dhcpv6_client_set_interface(&dhcp_0, index);` Set the Client network interface for communications with the DHCPv6 Server. |
| nx_dhcpv6_create_client_duid | `nx_dhcpv6_create_client_duid(&dhcp_0, NX_DHCPV6_DUID_TYPE_LINK_TIME, NX_DHCPV6_HW_TYPE_IEEE_802, 0)` Create a DHCPv6 Client DUID. |
| nx_dhcpv6_create_client_ia | `nx_dhcpv6_create_client_ia(&dhcp_0, &ipv6_address, NX_DHCPV6_PREFERRED_LIFETIME, NX_DHCPV6_VALID_LIFETIME);` Legacy Add a DHCPv6 Client Identity Address (IA). |
| nx_dhcpv6_create_client_iana | `nx_dhcpv6_create_client_iana(&dhcp_0, DHCPV6_IA_ID, DHCPV6_T1, DHCPV6_T2);` Create a DHCPv6 Client Identity Association for Non-Temporary Addresses (IANA). |

| Function Name | Example API Call and Description |
|---|---|
| nx_dhcpv6_add_client_ia | `nx_dhcpv6_add_client_ia(&dhcp_0, &ipv6_address, NX_DHCPV6_PREFERRED_LIFETIME, NX_DHCPV6_VALID_LIFETIME);` Add a DHCPv6 Client Identity Address (IA). |
| nx_dhcpv6_get_client_duid_time_id | `nx_dhcpv6_get_client_duid_time_id(&dhcp_0, &time_ID);` Get the time ID from DHCPv6 Client DUID. |
| nx_dhcpv6_get_ip_address | `nx_dhcpv6_get_IP_address(&dhcp_0, &ipv6_address);` `nxd_ipv6_address_set(&ip_0, 0, &ipv6_address, 64, &address_index);` Get the global IPv6 address assigned to the DHCPv6 client. |
| nx_dhcpv6_get_lease_time_data | `nx_dhcpv6_get_lease_time_data(&dhcp_0, &T1, &T2, &preferred_lifetime, &valid_lifetime);` Get T1 and T2 in the Identity Association (IANA) leased to the DHCPv6 Client. |
| nx_dhcpv6_get_iana_lease_time | `nx_dhcpv6_get_iana_lease_time(&dhcp_0, &T1, &T2);` Get T1, T2, valid and preferred lifetimes for the DHCPv6 Client IPv6 address by address index. |
| nx_dhcpv6_get_valid_ip_address_count | `nx_dhcpv6_get_valid_ip_address_count(&dhcp_0, &address_count);` This service retrieves the count of the Client's valid IPv6 addresses. A valid IPv6 address is bound (assigned) to the Client and registered with the IP instance. Also useful for determining if the DHPCv6 Client has reached the bound state. |
| nx_dhcpv6_get_valid_ip_address_lease_time | `nx_dhcpv6_get_valid_ip_address_lease_time(&dhcp_0, &ip_address, &preferred_lifetime, &valid_lifetime);` Get T1, T2, valid and preferred lifetimes for the DHCPv6 Client IPv6 address by address index. |
| nx_dhcpv6_get_DNS_server_address | `nx_dhcpv6_get_DNS_server_address(&dhcp_0, index, &server_address);` Get DNS Server address at the specified index into the DHCPv6 Client DNS server list. |
| nx_dhcpv6_get_other_option_data | `nx_dhcpv6_get_other_option_data(&dhcp_0, option_code, buffer);` Get the specified option data, such as domain name or time zone server. |
| nx_dhcpv6_get_time_accrued | `nx_dhcpv6_get_time_accrued(&dhcp_0, &time_accrued);` Get the time accrued the global IPv6 address lease has been bound to the DHCPv6 Client. |
| nx_dhcpv6_get_time_server_address | `nx_dhcpv6_get_time_server_address(&dhcp_0, index, &server_address);` Get Time Server address at the specified index into the DHCPv6 Client Time server list. |
| nx_dhcpv6_reinitialize | `nx_dhcpv6_reinitialize(&dhcp_0);` Reinitialize the DHCPv6 for restarting the DHCPv6 Client state machine and rerunning the DHCPv6 protocol. |
| nx_dhcpv6_request_confirm | `nx_dhcpv6_request_confirm(&dhcp_0);` Send a CONFIRM request to the Server. |
| nx_dhcpv6_request_inform_request | `nx_dhcpv6_request_inform_request(&dhcp_0);` Send an INFORM REQUEST message to the Server. |
| nx_dhcpv6_request_option_DNS_server | `nx_dhcpv6_request_option_DNS_server(&dhcp_0, NX_TRUE);` Add the DNS server option to the Client option request data in request messages to the Server. |

| Function Name | Example API Call and Description |
|---|---|
| nx_dhcpv6_request_option_FQDN | `nx_dhcpv6_request_option_FQDN(&dhcp_0, “DHCPv6_Client”, NX_DHCPV6_CLIENT_DESIRES_NO_SERVER_DNS_UPDATE);` Add the FQDN option to the Client option request data in request messages to the Server. |
| nx_dhcpv6_request_option_domain_name | `nx_dhcpv6_request_option_domain_name(&dhcp_0, NX_TRUE);` Add the domain name option to the Client option request data in request messages to the Server. |
| nx_dhcpv6_request_option_time_server | `nx_dhcpv6_request_option_time_server(&dhcp_0, NX_TRUE);` Add the time server option to the Client option request data in request messages to the Server. |
| nx_dhcpv6_request_option_timezone | `nx_dhcpv6_request_option_timezone(&dhcp_0, NX_TRUE);` Add the time zone option to the Client option request data in request messages to the Server. |
| nx_dhcpv6_request_release | `nx_dhcpv6_request_release(&dhcp_0);` Send a RELEASE request to the Server. |
| nx_dhcpv6_request_solicit | `nx_dhcpv6_request_solicit(&dhcp_0);` Send a DHCPv6 SOLICIT request to any Server on the Client network (broadcast). |
| nx_dhcpv6_request_solicit_rapid | `nx_dhcpv6_request_solicit_rapid(&dhcp_0);` Send a DHCPv6 SOLICIT request to any Server on the Client network (broadcast) with the Rapid Commit option set. |
| nx_dhcpv6_resume | `nx_dhcpv6_resume(&dhcp_0);` Resume DHCPv6 Client processing. |
| nx_dhcpv6_set_time_accrued | `nx_dhcpv6_set_time_accrued(&dhcp_0, time_accrued);` Set the time accrued on the global Client IPv6 address lease in the Client record. |
| nx_dhcpv6_start | `nx_dhcpv6_start(&dhcp_0);` Start the DHCPv6 Client thread task. Note this is not equivalent to starting the DHCPv6 state machine and does not send a SOLICIT request. |
| nx_dhcpv6_stop | `nx_dhcpv6_stop(&dhcp_0);` Stop the DHCPv6 Client thread task. |
| nx_dhcpv6_suspend | `nx_dhcpv6_suspend(&dhcp_0);` Suspend the DHCPv6 Client thread task. |
| *The following services are available if NX_DHCPV6_CLIENT_RESTORE_STATE is defined for the project:* | |
| nx_dhcpv6_client_get_record | `nx_dhcpv6_client_get_record(dhcpv6_ptr, client_record_ptr);` Obtain a record of the client state (to save to non-volatile memory) |
| nx_dhcpv6_client_restore_record | `nx_dhcpv6_client_restore_record(dhcpv6_ptr, client_record_ptr, time_elapsed);` Apply saved client record to the current Client instance. Note the DHCPv6 Client thread must be not be running when this task is called. |

Note: For details on operation and definitions for function data structures, typedefs, defines, API data, API structures and function variables review the associated Express Logic User’s Manual accessible as described in the References section at the end of this document.

**Table 4 Status Return Values**

| Name | Description |
|---|---|
| NX_SUCCESS | Successful API call. |
| NX_PTR_ERROR* | Invalid pointer input. |
| NX_CALLER_ERROR* | Must be called from thread. |
| NX_DHCPV6_PARAM_ERROR | Invalid non pointer input. |
| NX_INVALID_INTERFACE | Invalid interface index input. |
| NX_DHCPV6_UNSUPPORTED_DUID_TYPE | DUID type unknown or not supported. |
| NX_DHCPV6_UNSUPPORTED_DUID_HW_TYPE | DUID hardware type unknown or not supported. |
| NX_DHCPV6_IA_ADDRESS_ALREADY_EXIST | Duplicate IA address. |
| NX_DHCPV6_REACHED_MAX_IA_ADDRESS | IA exceeds the max IAs Client can store. |
| NX_DHCPV6_INVALID_IA_ADDRESS | Invalid (e.g. null) IA address in IA. |
| NX_DHCPV6_IA_ADDRESS_NOT_VALID | IPv6 address successfully assigned. |
| NX_DHCPV6_UNKNOWN_OPTION | Unknown/unsupported option code. |
| NX_DHCPV6_ALREADY_STARTED | DHCPv6 Client is already running. |
| NX_DHCPV6_NOT_STARTED | DHCPv6 Client task not started. |
| NX_DHCPV6_MISSING_REQUIRED_OPTIONS | Client missing required options. |

Note: Lower-level drivers may return common error codes. See *SSP User's Manual* API References for the associated module for a definition of all relevant status return values.
*These are error codes are only returned when error checking is enabled. See *NetX Duo User Guide for the Renesas Synergy™ Platform* for details on NetX Duo error-checking services.

## 3. NetX Duo DHCP Client Module Operational Overview

The DHCP Client handles obtaining an IP address, registering it with the IP instance, and renewing the IP address lease before the lease expires.

### DHCP IPv4 Client

A NetX Duo IP instance is created with a zero IP address and is enabled for UDP and ARP, respectively; RARP should not be enabled. A DHCP Client and UDP socket are created for sending and receiving DHCP messages. By default, the DHCP Client creates its own packet pool based on the settings *Minimum packet payload size* and *Number of packets in packet pool*. The *Minimum Client packet payload size* must be sufficiently large include DHCP data, IP, UDP headers, and the physical frame header.

- For Ethernet networks, this minimum payload is 592 bytes, which is the default setting of *Minimum Client packet payload size.*
- For other network types, such as Wi-Fi, the frame header size is larger, so the minimum size must be increased accordingly.

When the packet pool is created, the DHCP Client verifies the packet payload is not less than the minimum required payload size.

The DHCP Client can request a specific IP address using the `nx_dhcp_request_client_ip` service and supply a non-zero IP address before starting the DHCP Client. Normally, this is useful for a device that has been assigned an IP address previously and wishes to keep the same IP address. (The server is not obligated to accommodate this request.)

When the DHCP Client is started, it binds the socket to the DHCP port (by default 68) and begins sending and receiving packets through that socket. When the client is assigned an IP address, it automatically registers the IP address with NetX Duo. The server supplies the network mask and network gateway, and the DHCP Client updates NetX Duo with that information.

When the server assigns the client an IP address, it may also supply other network information, such as DNS server and NTP server. The application can obtain those values using the `nx_dhcp_user_option_retrieve` service.

The DHCP Client keeps track of time remaining on the IP lease; it automatically sends renew requests to the DHCP Server when it is time to renew. If the server is no longer on the network, or is otherwise not responding, the client sends broadcast requests to any DHCP Server on the network. If the lease expires without a renewal or rebinding, the client is returned to the NX_DHCP_STATE_INIT state. (The device may continue to use the IP address.) If a DHCP Server becomes available, and the device is able to request an IP address, it must no longer use the old IP address.

In busy networks, a DHCP Client socket queue can fill up with non-specific DHCP broadcast packets intended for other DHCP Client hosts. If the DHCP Client socket receive queue fills up, packets intended for the device may get dropped. To avoid this problem, the DHCP Client continually clears the socket of these non-specific broadcast packets.

## DHCP IPv6 Client

The DHCPv6 protocol also uses the UDP to dynamically obtain IPv6 addresses. A NetX Duo IP instance is created automatically, and UDP, IPv6, and ICMPv6 are enabled on the IP instance prior to creating the DHCPv6 Client. When the DHCPv6 Client is created, a UDP socket is created and bound to port 546. The *IPv6 Global Address* property of the IP instance is locked to a zero IPv6 address; otherwise, the IPv4 address may be any network IP address. If the IP instance *IPv6 Link Local Address* property is set to zero, the DHCPv6 Client creates the link local address from the MAC address. The MAC address is set in the NetX Port ETHER instance. See *Channel 1 MAC Address High Bits* and *Channel 1 MAC Address Low Bits* properties. This is the source address for DHCPv6 messages to the server.

To begin the process of requesting a global IPv6 address assignment, a Client first broadcasts a SOLICIT message using the `nx_dhcpv6_request_solicit` service. In IPv6, the broadcast address is the `All_DHCP_Relay_Agents_and_Servers` address (FF02::1:2.). A DHCPv6 Server responds with an ADVERTISE message containing a global IPv6 address (not a link local address) for the Client, the IPv6 address lease time, and any additional information requested by the client. The DHCPv6 protocol requires the client to wait for a period of time to receive ADVERTISE messages from all DHCPv6 Servers on the network. The client pre-processes each ADVERTISE message to be a valid message and scans the option data for various DHCPv6 parameters; it also checks the preference value in the preference option, if supplied by the server. If more than one ADVERTISE message is received, the NetX DHCPv6 Client chooses the ADVERTISE message with the highest preference value received by the end of the wait period. If the Client receives an ADVERTISE message with a preference value of 255, it accepts that message immediately and discards all subsequent ADVERTISE messages.

The client extracts data from the ADVERTISE message and broadcasts (so all DHCPv6 Servers are informed) a REQUEST message specifying which server the client chooses; that server then confirms the assigned address information and lease times with a REPLY message to complete the protocol.

The DHCPv6 Client is promoted to the bound state and automatically registers the assigned IPv6 address with the IP instance.

**Notification of Successful Address Assignment and Validation**

The application can determine when the DHCPv6 Client is bound to an IPv6 address in two ways: the first is to query the DHCPv6 Client itself for a valid IPv6 address using the `nx_dhcpv6_get_IP_address service` for clients for applications using only one global IPv6 address assigned, (the general case) or `nx_dhcpv6_get_valid_ip_address_count` service for clients with more than one IPv6 address assigned. The second way requires the DHCPv6 Client be configured with the state change callback, the *Name of state change notification function* property* of the DHCPv6 Client stack element.

If the DHCPv6 Client receives a response from the server, but the server is unable to assign the address, the server returns an error status. The application is notified of the error status received if configured with the DHCPv6 Client server-error callback — the *Name of server error handler* property of the DHCPv6 Client stack element.

These callbacks must be defined by the application because these callback functions are called from the DHCPv6 Client thread task; the client application must NOT call any NetX Duo DHCPv6 Client services that require mutex control of the DHCPv6 Client (such as `nx_dhcpv6_start`, `nx_dhcpv6_stop`), and any of the APIs that send messages directly from the callback (such as `nx_dhcpv6_request_release`).

The states of the DHCP IPv6 protocol are:

NX_DHCPV6_STATE_INIT
NX_DHCPV6_STATE_SENDING_SOLICIT
NX_DHCPV6_STATE_SENDING_REQUEST
NX_DHCPV6_STATE_SENDING_RENEW
NX_DHCPV6_STATE_SENDING_REBIND
NX_DHCPV6_STATE_SENDING_DECLINE
NX_DHCPV6_STATE_SENDING_INFORM_REQUEST
NX_DHCPV6_STATE_BOUND_TO_ADDRESS

**Duplicate Address Detection of the IPv6 address**

If configured for the Duplicate Address Detection (DAD) protocol, enabled by default in NetX Duo *Duplicate Address Detection property*, NetX Duo automatically sends "Neighbor Solicit" messages to verify the assigned address is unique on the network. If the IPv6 address is unique, NetX Duo notifies the DHCPv6 Client when the assigned address has been promoted from `NX_IPV6_ADDR_STATE_TENTATIVE to NX_IPV6_ADDR_STATE_VALID` internally. The application must allow time for the DAD to finish processing, which takes about 4-5 seconds. If the DAD is not enabled, the IP instance marks the address VALID immediately.

Once an IPv6 address is valid, the device may use that IPv6 address to send and transmit IPv6 messages.

If the DAD protocol fails, NetX Duo notifies the DHCPv6 Client to send a DECLINE message to the server and restart the DHCPv6 Client at the INIT state.

**Retransmission of DHCPv6 Client Solicitations**

The DHCPv6 Client times out waiting for a server reply before sending another DHCPv6 message and defaults to 1 second on the first retransmit, per RFC 3315 recommendations. If the DHCPv6 Client fails to receive a valid server response to the SOLICIT message, each subsequent retransmission interval doubles up to a maximum of 120 seconds by default.

**DHCPv6 Lease Timeouts**

The IPv6 lease assigned by the server contains two timeout parameters (T1 and T2) in the DHCPv6 Client Identity Association – Non-Temporary Addresses, (IANA), which is the data type specified by RFC 3315 to store IPv6 address data in DHCPv6.

When the time elapsed on an assigned IPv6 address reaches T1, the DHCPv6 Client automatically sends a RENEW message. If the elapsed time reaches T2 without a successful renewal, DHCPv6 Client automatically sends a REBIND message. If it still receives no response, the DHCP Client unregisters the IPv6 address with the IP instance and restarts the DHCPv6 protocol at the INIT state. Two other IPv6 lease parameters, preferred and valid lifetime, are assigned automatically to the Identity Association (IA) contained in the IANA in the DHCPv6 process. When the preferred and valid lifetimes expire, the assigned IPv6 address is either deprecated or rendered invalid; meaning a valid T1 must be less than the preferred lifetime and a T2 must be less than the valid lifetime.

## 3.1 NetX Duo DHCP Client Module Important Operational Notes and Limitations

### 3.1.1 NetX Duo DHCP Client Module Operational Notes

#### DHCP IPv4 Client

Instead of the DHCP Client Module creating the packet pool, the developer may prefer to supply a previously created packet pool. To so, enable the *Use application packet pool* option, then use the nx_dhcp_packet_pool_set service to set the DHCP Client's packet pool. (As previously described, the DHCP Client verifies that the packet payload is not less than the minimum required packet size.)

The IP address offered to the Client should be tested for 'uniqueness' on the local network since DHCP protocol does not require the server to do so. To configure the DHCP Client to perform this check, enable the *Send ARP probe* option. The DHCP Client sends a series of ARP "probes" with its assigned IP address out on the network; ff any host responds to these ARP requests/probes, the DHCP Client automatically sends a DECLINE message to the server and restarts the DHCP protocol to request another IP address. Otherwise, the DHCP Client proceeds to the bound state. The states of the client in the DHCP protocol are:

NX_DHCP_STATE_NOT_STARTED
NX_DHCP_STATE_INIT
NX_DHCP_STATE_SELECTING
NX_DHCP_STATE_REQUESTING
NX_DHCP_STATE_BOUND
NX_DHCP_STATE_RENEWING
NX_DHCP_STATE_REBINDING

Note: If an ARP probe is enabled, the NetX Duo DHCP Client enters a temporary state called `NX_DHCP_STATE_ADDRESS_PROBING` before the `NX_DHCP_STATE_BOUND` state.

The application can detect if the DHCP Client has completed (has an IP address) in a couple of ways: the first is to call the `nx_ip_status_check` service with the NX_IP_ADDRESS_RESOLVED option. An alternative is to use the `nx_dhcp_state_change_notify` service which notifies the application of DHCP Client state changes. When the DHCP Client reaches the bound state, (state == NX_DHCP_STATE_BOUND) it has a valid IP address.

There may be a need to stop the DHCP Client thread task; to do so, call the `nx_dhcp_stop` service. To restart the client, first call the `nx_dhcp_reinitialize` service to clear the DHCP Client data and clear network parameters registered with NetX Duo; the DHCP Client is then restarted with the `nx_dhcp_start` call.

### DHCP IPv6 Client

- The NetX Duo Source element has a few key properties for supporting DHCPv6: the *NetX Duo IPv6 Support* property, the *Checksum computation support on received ICMPV6 packets,* and the *Checksum computation support on transmitted ICMPv6 packets*. The latter two ensure that incoming and outgoing packets have an ICMPv6 checksum in the ICMPv6 header; these are automatically enabled for the DHCPv6 Client module. If the NetX Duo Source element is added to the project, check to make sure these properties are enabled.
- The DHCPv6 Client creation requires a previously created packet pool. The application can use the IP default packet pool (`g_packet_pool0`) used by the IP instance or it can create its own (usually `g_packet_pool1`.)
- Before sending a SOLICIT request, the Client must create a DHCP Unique Identifier (DUID) to uniquely define the client on the network. The MAC address is usually used but can be another unique identifier. A typical invocation of this service is:

```
nx_dhcpv6_create_client_duid(&g_dhcpv6_client0,
                  NX_DHCPV6_DUID_TYPE_LINK_TIME /* Use MAC address */,
                NX_DHCPV6_HW_TYPE_IEEE_802,

                0 /* Client DUID time, usually set to zero */);
```

- The DHCPv6 Client must also create the IANA for the client; this structure holds IPv6 lease information such as IPv6 addresses and T1 and T2 times. (The client can use the IANA to request lease times.) To create an IANA, use the `nx_dhcpv6_client_create_iana` service:

```
status = nx_dhcpv6_create_client_iana(&g_dhcpv6_client0,

      DHCPV6_IANA_ID /* ULONG unique ID */,

      DHCPV6_T1 /* Request T1 time in seconds or

                 set to TX__WAIT_FOREVER */,

      DHCPV6_T2 /* Request T2 time;must be longer

                than T1 */);
```

- In the SOLICIT request, the client may request the assignment of a specific IPv6 address from the server by calling the `nx_dhcpv6_add_client_ia` service before calling `nx_dhcpv6_request_solicit`. This service uses an NXD_ADDRESS address data type for the IPv6 address. See *NetX Duo User Guide for the Renesas Synergy™ Platform* for details on the data type definition in NetX Duo.
- To request network information such as a DNS server, NTP server, and other options, the application can all these APIs before calling `nx_dhcpv6_request_solicit`:
  — `nx_dhcpv6_request_option_timezone(&g_dhcpv6_client, NX_TRUE);`
  — `nx_dhcpv6_request_option_DNS_server(&g_dhcpv6_client, NX_TRUE);`
  — `nx_dhcpv6_request_option_time_server(&g_dhcpv6_client, NX_TRUE);`
  — `nx_dhcpv6_request_option_domain_name(&g_dhcpv6_client, NX_TRUE);`

- If the client needs to release an assigned IPv6 address, it informs the DHCPv6 server by calling the `nx_dhcpv6_request_release` service. The DHCPv6 Client sends a unicast RELEASE message to the server and should wait for the server REPLY.
- For DHCPv6 Client services that retrieve information about the DHCPv6 Client, an address index may need to be specified. Most clients have one IPv6 global address assigned, so the address index is 0.
- To obtain specific information about lease times, use the `nx_dhcpv6_get_valid_ip_address_lease_time` service. This requires an address index input (usually 0.)

### 3.1.2 NetX Duo DHCP Client Module Limitations

**DHCP IPv4 Client**

- The DHCP Client does not support the INFORM_REQUEST message. The application can send this message out using the `nx_dhcp_send_request` service, but the data from the server is not extracted and saved to the DHCP Client.
- The options supported by `nx_dhcp_user_option_retrieve` are limited to the following:
  NX_DHCP_OPTION_SUBNET_MASK
  NX_DHCP_OPTION_TIME_OFFSET
  NX_DHCP_OPTION_GATEWAYS
  NX_DHCP_OPTION_TIMESVR
  NX_DHCP_OPTION_DNS_SVR
  NX_DHCP_OPTION_NTP_SVR
  NX_DHCP_OPTION_DHCP_LEASE
  NX_DHCP_OPTION_DHCP_SERVER
  NX_DHCP_OPTION_RENEWAL
  NX_DHCP_OPTION_REBIND
- For additional information on limitations, see *NetX Duo Dynamic Host Configuration Protocol for Clients User Guide for the Renesas Synergy™ Platform*, listed in the References section.
- For additional module operational limitations, see the latest SSP Release Notes.

**DHCP IPv6 Client**

- The NetX Duo DHCPv6 Client does not support the server unicast option for sending unicast DHCPv6 messages to the DHCPv6 Server even if the server indicates this is permitted.
- The NetX Duo DHCPv6 Client only supports DUIDs for LINK (MAC address) and LINK TIME (MAC address and time input.)
- The NetX Duo DHCPv6 Client does not support the reconfigure request in which a server initiates IPv6 address changes to the clients on the network.
- The NetX Duo DHCPv6 Client does not support the enterprise format for the DHCPv6 unique identifier control block; it only supports Link Layer and Link Layer Plus Time formats.
- The NetX Duo DHCPv6 Client does not support Temporary Association (TA) address requests, but does support Non Temporary (IANA) option requests.

For additional module operational limitations, see the latest SSP Release Notes.

# 4. Including the NetX Duo DHCP Client Module in an Application

This section describes how to include the NetX Duo DHCP Client Module in an application using the SSP configurator.

Note: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread, and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the SSP User's Manual to learn how to manage each of these important steps in creating SSP-based applications.

**DHCP IPv4 Client**

To add the NetX Duo DHCP IPv4 Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the NetX Duo DHCP Client is g_dhcp_client0. This name can be changed in the associated Properties window.)

**Table 5 NetX Duo DHCP IPv4 Client Module Selection Sequence**

| Resource | ISDE Tab | Stacks Selection Sequence |
|---|---|---|
| g_dhcp_client0 NetX Duo DHCP Client | Threads | New Stack> X-Ware> NetX> Protocols> NetX Duo DHCP Client |

The following figure shows when the NetX Duo DHCP IPv4 Client module is added to the thread stack the configurator automatically adds any required lower-level NetX Duo components and drivers. A driver or NetX Duo component needing additional configuration information has box text highlighted in red. Modules with a gray band are individual, standalone modules. Modules with a blue band are shared or common and added only once to be used by multiple stacks. Modules with a pink band can require the selection of lower-level drivers; sometimes these are optional or recommended as text in the block indicates. If additional lower-level drivers are required, the module description includes "Add" in the text. Click any pink-banded modules to display the "New" icon and show possible choices.

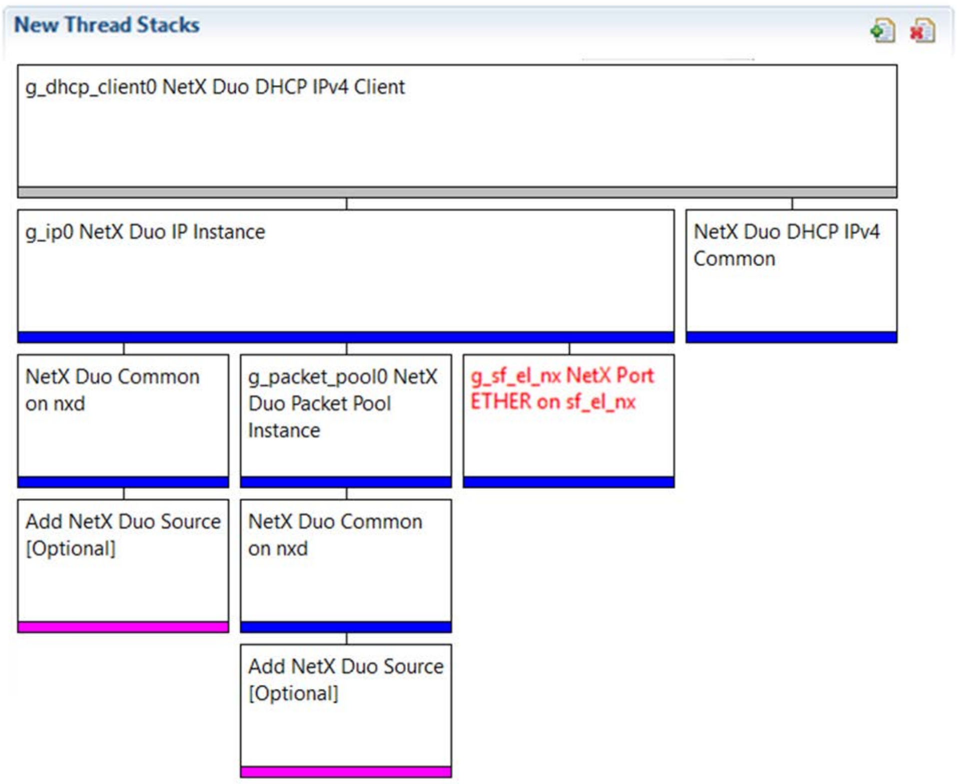
New Thread Stacks
g_dhcp_client0 NetX Duo DHCP IPv4 Client
g_ip0 NetX Duo IP Instance
NetX Duo DHCP IPv4 Common
NetX Duo Common on nxd
g_packet_pool0 NetX Duo Packet Pool Instance
g_sf_el_nx NetX Port ETHER on sf_el_nx
Add NetX Duo Source [Optional]
NetX Duo Common on nxd
Add NetX Duo Source [Optional]

**Figure 3 NetX Duo DHCP IPv4 Client Module Stack**

### DHCP IPv6 Client

To add the NetX Duo DHCPv6 Client module to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the NetX Duo DHCPv6 Client module is `g_dhcpv6_client0`. This name can be changed in the associated Properties window.)

**Table 6 NetX DHCP IPv6 Client Module Selection Sequence**

| Resource | ISDE Tab | Stacks Selection Sequence |
|---|---|---|
| g_dhcpv6_client0 NetX DHCPv6 Client | Threads | New Stack > Xware > NetX Duo -> Protocols > NetX Duo DHCP IPv6 Client |

When the NetX Duo DHCPv6 Client is added to the thread stack as shown in the following figure, the configurator automatically adds the needed lower-level drivers. Any drivers needing additional configuration information are box text highlighted in red. Modules with a gray band are individual, standalone modules. Modules with a blue band are shared or common and only added once to be used by multiple stacks. Modules with a pink band can require lower-level drivers; sometimes these are optional or recommended as text in the box indicates. If additional lower-level drivers are required, the module description includes "Add" in the text. Click any pink-banded modules to display the "New" icon and show possible choices. .

In the following figure, the DHCPv6 Client is configured with its own packet pool, `g_packet_pool1`. To adjust the properties of NetX Duo relating to IPv6 and ICMPv6, the underlying protocols for DHCPv6, select the **Add NetX Duo Source** stack element (either box) and choose **New** -> **NetX Duo Source**.

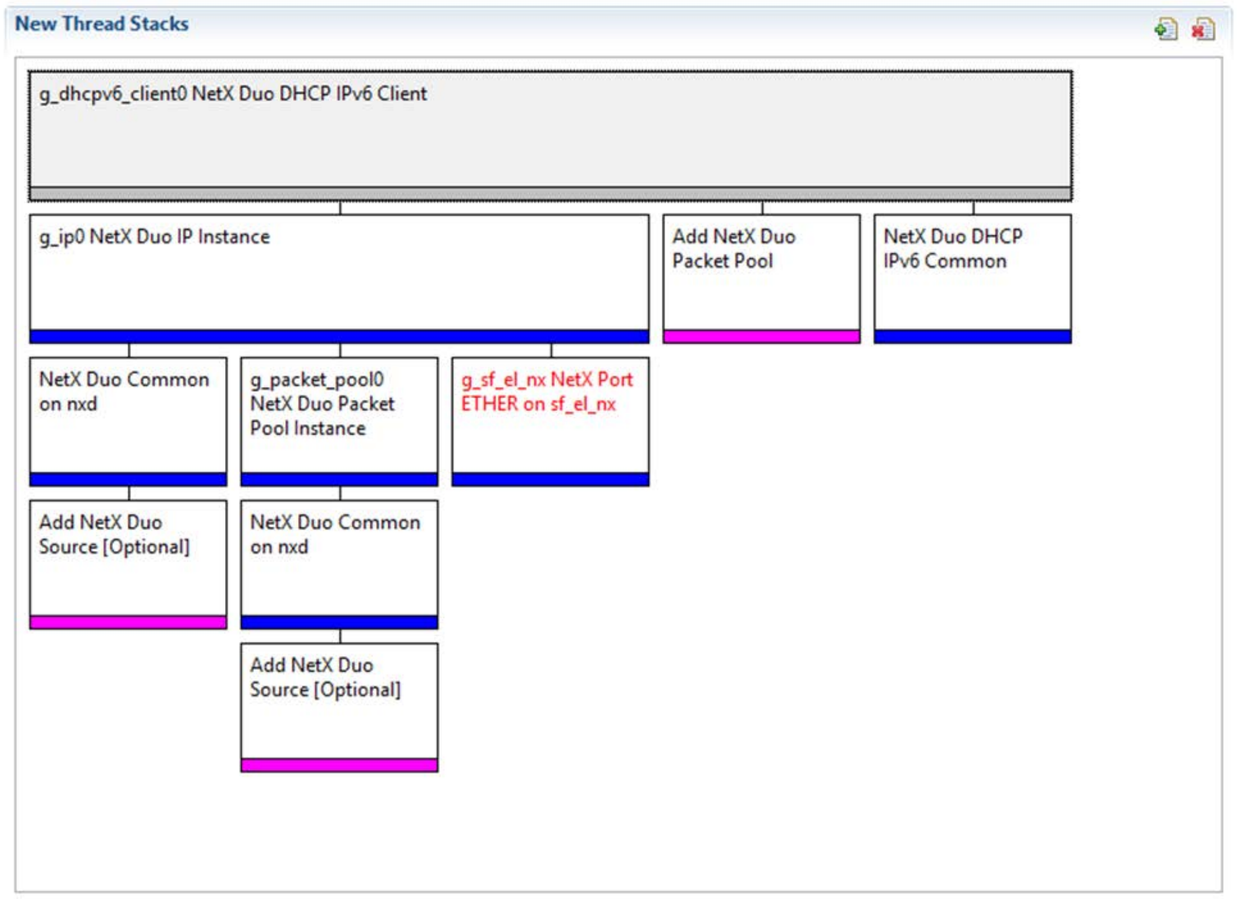
New Thread Stacks
g_dhcpv6_client0 NetX Duo DHCP IPv6 Client
g_ip0 NetX Duo IP Instance
Add NetX Duo Packet Pool
NetX Duo DHCP IPv6 Common
NetX Duo Common on nxd
g_packet_pool0 NetX Duo Packet Pool Instance
g_sf_el_nx NetX Port ETHER on sf_el_nx
Add NetX Duo Source [Optional]
NetX Duo Common on nxd
Add NetX Duo Source [Optional]

**Figure 4 NetX Duo DHCP IPv6 Client Module Stack**

## 5. Configuring the NetX Duo DHCP Client Module

The user configures the NetX Duo DHCP Client module for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) the selections required for successful operation, such as interrupts or operating modes for lower-level modules. Only non-conflicting properties are available for change. Unavailable properties are ‘locked’ with a lock icon in the ISDE Properties window. This approach simplifies the configuration process and makes it much less error-prone than previous ‘manual’ approaches. The SSP Configurator Properties tab shows available configuration settings and defaults for all the user-accessible properties. They are also listed in the following tables for easy reference.

Interrupt priorities are one of the properties often requiring change and the settings are available within the Properties window of the associated module. Simply select the indicated module and view the Properties window; the interrupt settings are toward the bottom of the properties list, so scroll down until they become available. Also note the interrupt priorities listed in the ISDE Properties window indicates the setting’s validity based on the targeted MCU (CM4 or CM0+). These details are not included in the following tables, but are easily visible in the ISDE when configuring interrupt-priority levels.

Note: You may want to open your ISDE, create the module, and explore the property settings in parallel with reviewing the following configuration table settings. This helps to orient you and can be a useful ‘hands-on’ approach to learning the ins and outs of developing with SSP.

### DHCP IPv4 Client

**Table 7 Configuration Settings for the NetX Duo DHCP IPv4 Client Module**

| Parameter | Value | Description |
|---|---|---|
| Internal thread priority | 3 | DHCP Client thread priority (should be lower priority than IP thread) |
| Internal thread stack size (bytes) | 4096 | DHCP Client thread stack size (bytes) |

| **Parameter** | **Value** | **Description** |
|---|---|---|
| Timeout between DHCP messages processed (seconds) | 1 | DHCP Client thread task sleep interval (seconds) |
| Packet allocate timeout (seconds) | 1 | Packet allocate timeout (seconds) selection |
| Use BOOTP | Enable, Disable<br>Default: Disable | Run DHCP with the BOOTP option |
| Send ARP probe | Enable, Disable<br>Default: Disable | Send ARP probe to verify unique IP address |
| Packet ARP probe timeout (seconds) | 10 | Packet ARP probe timeout (seconds) selection |
| Maximum retransmission timeout (seconds) | 64 | Maximum retransmission timeout (seconds) for resending DHCP messages to Server |
| Minimum renew timeout (seconds) | 60 | Timeout on renewal retransmissions decreases by half down to the Minimum renew timeout (seconds) |
| Minimum retransmission timeout (seconds) | 4 | Minimum retransmission timeout (seconds) is the starting timeout. This doubles up to the maximum retransmission timeout. |
| Client packet payload size (bytes) | 592 | Minimum DHCP Client packet payload size (bytes) to include all frame and network headers plus DHCP data. |
| Number of packets in internal packet pool | 5 | Number of packets in DHCP created packet pool; no effect if Use application packet pool enabled |
| Server message check interval (unit = processing interval). Disabled = 0xFFFFFFF | 0xFFFFFFF | DHCP Client checks for Server message in the Bound state every (check interval * sleep interval seconds) |
| Persistent client state | Enable, Disable<br>Default: Disable | Enable DHCP Client to be restored after power is cycled on the device |
| Clear queue packets support | Enable, Disable<br>Default: Disable | Clear queue packets support selection |
| Use application packet pool | Enable, Disable<br>Default: Disable | Supply an application created packet pool to the DHCP Client; number of packets in packet pool and Minimum packet payload size have no effect |
| Maximum message size support | Enable, Disable<br>Default: Disable | DHCP Client specified maximum message size to the Server |
| DHCP options buffer size (bytes) | 312 | DHCP options buffer size (bytes) selection |
| ARP probe wait time (seconds) | 1 | How long to wait for response to an ARP probe for uniqueness of assigned IP address |
| Minimum ARP probe wait time (seconds) | 1 | A random delay is added to the ARP wait time; ensure the wait option is never less than 1 second. |
| Maximum ARP probe wait time (seconds) | 2 | A random delay is added to the ARP wait time; ensure the wait option is never more than 2 seconds. |
| ARP probe count | 2 | Number of ARP probes sent after receiving a DHCP Server IP address lease. |
| Name | g_dhcp_client0 | Module name |

Note: The example settings and defaults are for a project using the Synergy S7G2 Family. Other MCUs may have different default values and available configuration settings.

### DHCP IPv6 Client

**Table 8 Configuration Settings for the DHCP IPv6 Client Module**

| Parameter | Value | Description |
|---|---|---|
| Internal thread priority | 3 | Internal thread priority selection |
| Time out for obtaining DHCPv6 client mutex (ticks) | TX_WAIT_FOREVER | Internal time out for obtaining DHCPv6 Client mutex. This permits both the DHCPv6 Client thread task and application to call DHCPv6 Client services without conflict |
| Time interval between current IP address lease time update (seconds) | 1 | Time slice when DHCPv6 Client thread task performs periodic tasks |
| Maximum IA addresses allowed in client record" | 1 | Equivalent to the number of IPv6 addresses the DHCPv6 Client is configured to have. |
| Number of DNS servers the client stores | 2 | Max number of DNS server IPv6 addresses stored in the DHCPv6 Client block |
| Number of time servers the client stores | 1 | Max number of time server IPv6 addresses stored in the DHCPv6 Client block |
| Domain name buffer size (bytes) | 32 | Size of buffer to hold the network domain name supplied by the DHCPv6 Server |
| Current time zone information buffer size (bytes) | 10 | Size of buffer to hold the time zone, such as EST, PST that is supplied by the DHCPv6 Server |
| Maximum DHCPv6 server messages buffer size (bytes) | 100 | Size of buffer to hold the entire DHCPv6 Server message |
| Name | g_dhcpv6_client0 | Name of the DHCPv6 Client instance |
| Internal Thread stack size (bytes) | 4096 | Size of stack memory for the DHCPv6 Client |
| Name of state change notification function | dhcpv6_state_change_notify | Callback function for DHCPv6 Client to notify the application when the Client has changed DHCPv6 state |
| Name of server error handler | dhcpv6_server_error_handler | Callback function for DHCPv6 Client to forward status code, DHCPv6 parameters and DHCPv6 message type if the server returns an error status |

Note: The example settings and defaults are for a project using the Synergy S7G2 Family. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different Ethernet interface pins and resets. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference. Most of the property settings for lower-level modules are fairly intuitive and usually can be determined by inspection of the associated Properties window from the SSP configurator.

## 5.1 Configuration Settings for the NetX DHCP Client Lower-Level Modules

Typically, only a few settings (indicated by red text in the thread stack block) need modification from their default for the IP layer and lower-level drivers. Notice which configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following tables identify all the settings within the properties section for the module.

### DHCP IPv4 Client

**Table 9 Configuration Settings for the NetX Duo IP Instance**

| ISDE Property | Value | Description |
|---|---|---|
| Name | g_ip0 | Module name |
| IPv4 Address (use commas for separation) | 0,0,0,0 | IPv4 Address selection |
| Subnet Mask (use commas for separation) | 255,255,255,0 | Subnet Mask selection |
| IP Helper Thread Stack Size (bytes) | 2048 | IP Helper Thread Stack Size (bytes) selection |
| IP Helper Thread Priority | 3 | IP Helper Thread Priority selection |
| ARP | Enable | ARP selection |
| ARP Cache Size in Bytes | 512 | ARP Cache Size in Bytes selection |
| Reverse ARP | Enable, Disable<br>Default: Disable | Reverse ARP selection |
| TCP | Enable, Disable<br>Default: Enable | TCP selection |
| UDP | Enable | UDP selection |
| ICMP | Enable, Disable<br>Default: Enable | ICMP selection |
| IGMP | Enable, Disable<br>Default: Enable | IGMP selection |
| IP fragmentation | Enable, Disable<br>Default: Disable | IP fragmentation selection |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 10 Configuration Settings for the NetX Duo DHCP IPv4 Common Instance**

| ISDE Property | Value | Description |
|---|---|---|
| Type of Service for UDP requests | Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost<br>Default: Normal | Type of service UDP requests selection |
| Fragmentation option | Don't fragment, Fragment okay<br>Default: Don't fragment | Fragment option selection |
| Time to live | 128 | Time to live selection |
| Packet Queue depth | 5 | Packet queue depth selection |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 11 Configuration Settings for the NetX Duo Common Instance**

| ISDE Property | Value | Description |
|---|---|---|
| No configurable settings | | |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 12 Configuration Settings for the NetX Duo Packet Pool Instance**

| ISDE Property | Value | Description |
|---|---|---|
| Name | g_packet_pool0 | Module name |
| Packet Size in Bytes | 640 | Packet size selection |
| Number of Packets in Pool | 16 | Number of packets in pool selection |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 13 Configuration Settings for the NetX Port ETHER**

| ISDE Property | Value | Description |
|---|---|---|
| Parameter Checking | BSP, Enabled, Disabled<br>Default: BSP | Enable or disable the parameter checking |
| Channel 0 Phy Reset Pin | IOPORT_PORT_09_PIN_03 | Channel 0 Phy reset pin selection |
| Channel 0 MAC Address High Bits | 0x00002E09 | Channel 0 MAC address high bits selection |
| Channel 0 MAC Address Low Bits | 0x0A0076C7 | Channel 0 MAC address low bits selection |
| Channel 1 Phy Reset Pin | IOPORT_PORT_07_PIN_06 | Channel 1 Phy reset pin selection |
| Channel 1 MAC Address High Bits | 0x00002E09 | Channel 1 MAC address high bits selection |
| Channel 1 MAC Address Low Bits | 0x0A0076C8 | Channel 1 MAC address low bits selection |
| Number of Receive Buffer Descriptors | 8 | Number of receive buffer descriptors selection |
| Number of Transmit Buffer Descriptors | 32 | Number of transmit buffer descriptors selection |
| Ethernet Interrupt Priority | Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid)<br>Default: Disabled | Ethernet interrupt priority selection |
| Name | g_sf_el_nx | Module name |
| Channel | 0 | Channel selection |
| Callback | NULL | Callback selection |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 14 Configuration Settings for the NetX Duo Common Instance**

| ISDE Property | Value | Description |
|---|---|---|
| No configurable settings | | |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

### DHCP IPv6 Client

**Table 15 Configuration Settings for the NetX Duo IP Instance**

| ISDE Property | Value | Description |
|---|---|---|
| Name | g_ip0 | Module name |
| IPv4 Address (use commas for separation) | 192,168,0,2 | IPv4 Address selection |
| Subnet Mask (use commas for separation) | 255,255,255,0 | Subnet Mask selection |
| **IPv6 Global Address (use commas for separation) | 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0 | IPv6 global address selection |
| **IPv6 Link Local Address (use commas for separation, All zeros means use MAC address) | 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0 | IPv6 link local address selection |
| IP Helper Thread Stack Size (bytes) | 2048 | IP Helper Thread Stack Size (bytes) selection |
| IP Helper Thread Priority | 3 | IP Helper Thread Priority selection |
| ARP | Enable | ARP selection |
| ARP Cache Size in Bytes | 512 | ARP Cache Size in Bytes selection |
| Reverse ARP | Enable, Disable<br>Default: Disable | Reverse ARP selection |
| TCP | Enable, Disable<br>Default: Enable | TCP selection |
| UDP | Enable | UDP selection |
| ICMP | Enable, Disable<br>Default: Enable | ICMP selection |
| IGMP | Enable, Disable<br>Default: Enable | IGMP selection |
| IP fragmentation | Enable, Disable<br>Default: Disable | IP fragmentation selection |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 16 Configuration Settings for the NetX Duo Packet Pool Instance**

| ISDE Property | Value | Description |
|---|---|---|
| Name | g_packet_pool1 | Module name |
| Packet Size in Bytes | 640 | Packet size selection |
| Number of Packets in Pool | 16 | Number of packets in pool selection |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 17 Configuration Settings for the NetX Duo DHCP IPv6 Common Instance**

| ISDE Property | Value | Description |
|---|---|---|
| Type of Service for UDP requests | Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost<br>Default: Normal | Type of service UDP requests selection |
| Time to live | 128 | Time to live selection |
| Packet Queue depth | 5 | Packet queue depth selection |

| ISDE Property | Value | Description |
|---|---|---|
| packet allocation timeout (seconds) | 3 | Packet allocation timeout selection |
| Interval for active session time update (seconds) | 3 | Interval for active session time update selection |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 18 Configuration Settings for the NetX Duo Common Instance**

| ISDE Property | Value | Description |
|---|---|---|
| No configurable settings | | |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 19 Configuration Settings for the NetX Duo Packet Pool Instance**

| ISDE Property | Value | Description |
|---|---|---|
| Name | g_packet_pool0 | Module name |
| Packet Size in Bytes | 640 | Packet size selection |
| Number of Packets in Pool | 16 | Number of packets in pool selection |

**Table 20 Configuration Settings for the NetX Port ETHER**

| ISDE Property | Value | Description |
|---|---|---|
| Parameter Checking | BSP, Enabled, Disabled<br>Default: BSP | Enable or disable the parameter checking |
| Channel 0 Phy Reset Pin | IOPORT_PORT_09_PIN_03 | Channel 0 Phy reset pin selection |
| Channel 0 MAC Address High Bits | 0x00002E09 | Channel 0 MAC address high bits selection |
| Channel 0 MAC Address Low Bits | 0x0A0076C7 | Channel 0 MAC address low bits selection |
| Channel 1 Phy Reset Pin | IOPORT_PORT_07_PIN_06 | Channel 1 Phy reset pin selection |
| Channel 1 MAC Address High Bits | 0x00002E09 | Channel 1 MAC address high bits selection |
| Channel 1 MAC Address Low Bits | 0x0A0076C8 | Channel 1 MAC address low bits selection |
| Number of Receive Buffer Descriptors | 8 | Number of receive buffer descriptors selection |
| Number of Transmit Buffer Descriptors | 32 | Number of transmit buffer descriptors selection |
| Ethernet Interrupt Priority | Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid)<br>Default: Disabled | Ethernet interrupt priority selection |
| Name | g_sf_el_nx | Module name |
| Channel | 0 | Channel selection |
| Callback | NULL | Callback selection |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

**Table 21 Configuration Settings for the NetX Duo Common Instance**

| ISDE Property | Value | Description |
|---|---|---|
| No configurable settings | | |

Note: The example settings and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

## 5.2 NetX Duo DHCP Client Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set by using the SSP configurator clock tab prior to a build, or by using the CGC Interface at run-time.

## 5.3 NetX Duo DHCP Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I2C pins.

Note: The operation mode selection determines what peripheral signals are available and thus what MCU pins are required.

**Table 22 Pin Selection for the ETHERC Module**

| Resource | ISDE Tab | Pin selection Sequence |
|---|---|---|
| ETHERC | Pins | Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII |

Note: The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

**Table 23 Pin Configuration Settings for the ETHERC1**

| Property | Value | Description |
|---|---|---|
| Operation Mode | Disabled, Custom, RMII (Default: Disabled) | Select RMII as the Operation Mode for ETHERC1 |
| Pin Group Selection | Mixed, _A only (Default: _A only) | Pin group selection |
| REF50CK | P701 | REF50CK Pin |
| TXD0 | P700 | TXD0 Pin |
| TXD1 | P406 | TXD1 Pin |
| TXD_EN | P405 | TXD_EN Pin |
| RXD0 | P702 | RXD0 Pin |
| RXD1 | P703 | RXD1 Pin |
| RX_ER | P704 | RX_ER Pin |
| CRS_DV | P705 | CRS_DV Pin |
| MDC | P403 | MDC Pin |
| MDIO | P404 | MDIO Pin |

Note: The example settings are for a project using the Synergy S7G2 and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

# 6. Using the NetX Duo DHCP Client Module in an Application

### DHCP IPv4 Client

The following example assumes that a system is already established with a working IP, the ARP and UDP are enabled, and the link is running. The typical steps in using the NetX Duo DHCP Client module in an application are:

1. Set DHCP features for the DHCP Client (request specific IP address, clear the broadcast flag, set the interface on which the DHCP Client runs) before starting the DHCP Client. [Optional]
2. Start the DHCP with the `nx_dhcp_start` API
3. Wait for IP address resolution by calling `nx_ip_status_check` (a NetX library service call) or check for the bound state in the DHCP Client state change callback function.
4. A valid IP address is now on lease and the application can now start using NetX services for sending and receiving packets.

5. The DHCP Client automatically requests IP lease renewal based on the time remaining on the IP lease (as long as the DHCP Client thread task is still running.) [Optional]
6. To stop the DHCP Client thread task, call `nx_dhcp_stop`. To restart the DHCP Client, call `nx_dhcp_reinitialize` and then call `nx_dhcp_start`. [Optional]
7. Add or modify existing DHCP Client settings. [Optional]

These common steps are illustrated in a typical operational flow diagram in the following figure:

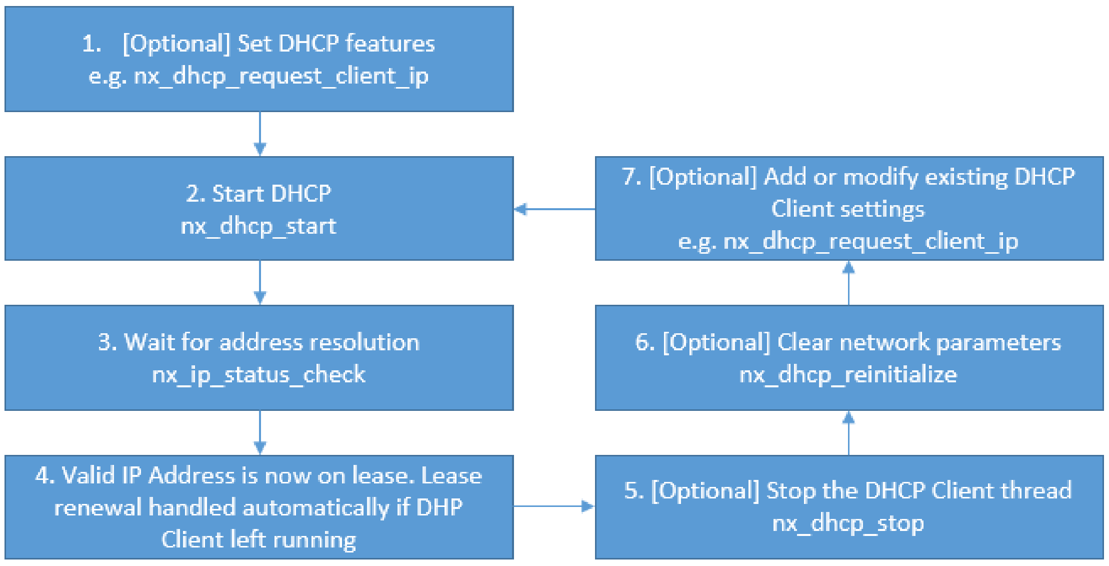
1. [Optional] Set DHCP features e.g. nx_dhcp_request_client_ip
2. Start DHCP nx_dhcp_start
3. Wait for address resolution nx_ip_status_check
4. Valid IP Address is now on lease. Lease renewal handled automatically if DHP Client left running
5. [Optional] Stop the DHCP Client thread nx_dhcp_stop
6. [Optional] Clear network parameters nx_dhcp_reinitialize
7. [Optional] Add or modify existing DHCP Client settings e.g. nx_dhcp_request_client_ip

**Figure 5 Typical NetX Duo DHCP IPv4 Client Module application flow**

### DHCP IPv6 Client

The following example assumes that a system is already establish with a working IP, ARP and UDP are enabled, and the link is running. The typical steps in using the NetX Duo DHCP Client module in an application are:

1. Create a Client DUID for the DHCPv6 Client using the `nx_dhcpv6_create_client_duid`.
2. Create an IANA for the DHCPv6 Client using the `nx_dhcpv6_create_client_iana`.
3. Request network options (such as DNS server and time server) using the `nx_dhcpv6_request_option_DNS_server` and `nx_dhcpv6_request_option_time_server` API `[Optional]`.
4. Start the DHCPv6 Client with the `nx_dhcpv6_start` API. This sets the DHCPv6 Client state and binds the client socket, in effect getting the DHCPv6 Client ready to exchange messages with the server.
5. Send the SOLICIT message to the server using the `nx_dhcpv6_request_solicit` API. The DHCPv6 Client internally manages the rest of the DHCPv6 protocol.
6. Check for IPv6 Address resolution using the `nx_dhcpv6_get_valid_ip_address_count` API to return with an `address_count` > 0. The DHCPv6 Client now as a valid IPv6 address.

The following figure shows the common steps in a typical operational flow.

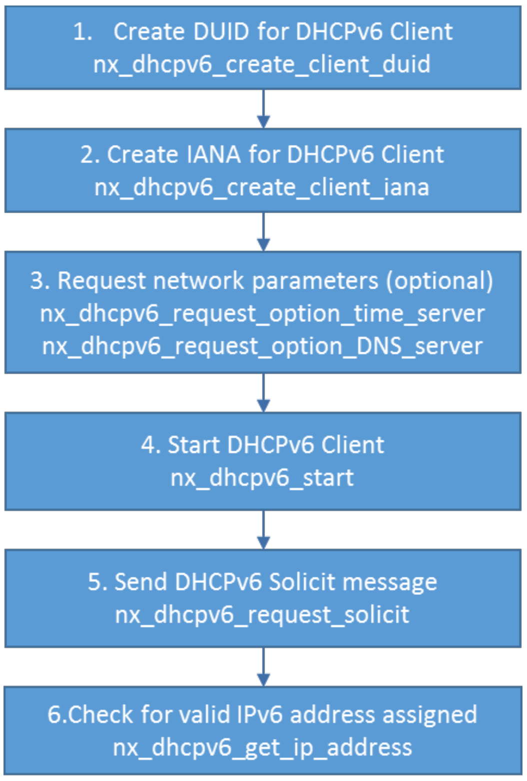
1. Create DUID for DHCPv6 Client
nx_dhcpv6_create_client_duid
2. Create IANA for DHCPv6 Client
nx_dhcpv6_create_client_iana
3. Request network parameters (optional)
nx_dhcpv6_request_option_time_server
nx_dhcpv6_request_option_DNS_server
4. Start DHCPv6 Client
nx_dhcpv6_start
5. Send DHCPv6 Solicit message
nx_dhcpv6_request_solicit
6.Check for valid IPv6 address assigned
nx_dhcpv6_get_ip_address

**Figure 6 Typical NetX Duo DHCP IPv6 Client Module application flow**

## 7. The NetX Duo DHCP Client Module Application Project

### DHCP IPv4 Client

The application project associated with this module guide demonstrates the aforementioned steps in a full design. The project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the NetX Duo DHCP IPv4 Client module. You can also read over the code (in `DHCPv4_Client_Netxduo_MG.c`) which is used to illustrate the NetX Duo DHCP IPv4 Client module APIs in a complete design.

This application project demonstrates the typical use of the NetX Duo DHCP IPv4 Client module APIs. It configures the DHCP Client with some common optional features before starting the DHCP Client thread task. It waits to be assigned a valid IP address and subsequently tests the IP address by pinging on the DHCP and DNS servers.

The `DHCPv4_Client_Netxduo_MG.c` file is located in the project once it has been imported into the ISDE. (This is not the auto generated code file `dhcp_thread_entry.c`.) Inside the `dhcp_thread_entry`, there is a call to `run_dhcp_client_session`, a function defined in the `DHCPv4_Client_Netxduo_MG.c` file to run a DHCP Client session. You can open `DHCPv4_Client_Netxduo_MG.c` within the ISDE and follow along with the following description to help identify key uses of APIs.

The following table identifies the target versions for the associated software and hardware used by the application project:

**Table 24 Software and Hardware Resources Used by the Application Project**

| Resource | Revision | Description |
|---|---|---|
| e$^2$ studio | 5.3.1 | Integrated Solution Development Environment |
| SSP | 1.2.0 | Synergy Software Platform |
| IAR EW for Renesas Synergy | 7.71.1 | IAR Embedded Workbench® for Renesas Synergy™ |
| SSC | 5.3.1.002 | Synergy Standalone Configurator |
| SK-S7G2 | v3.0 to v3.1 | Starter Kit |

A simple flow diagram of the application project is given in the figure below:

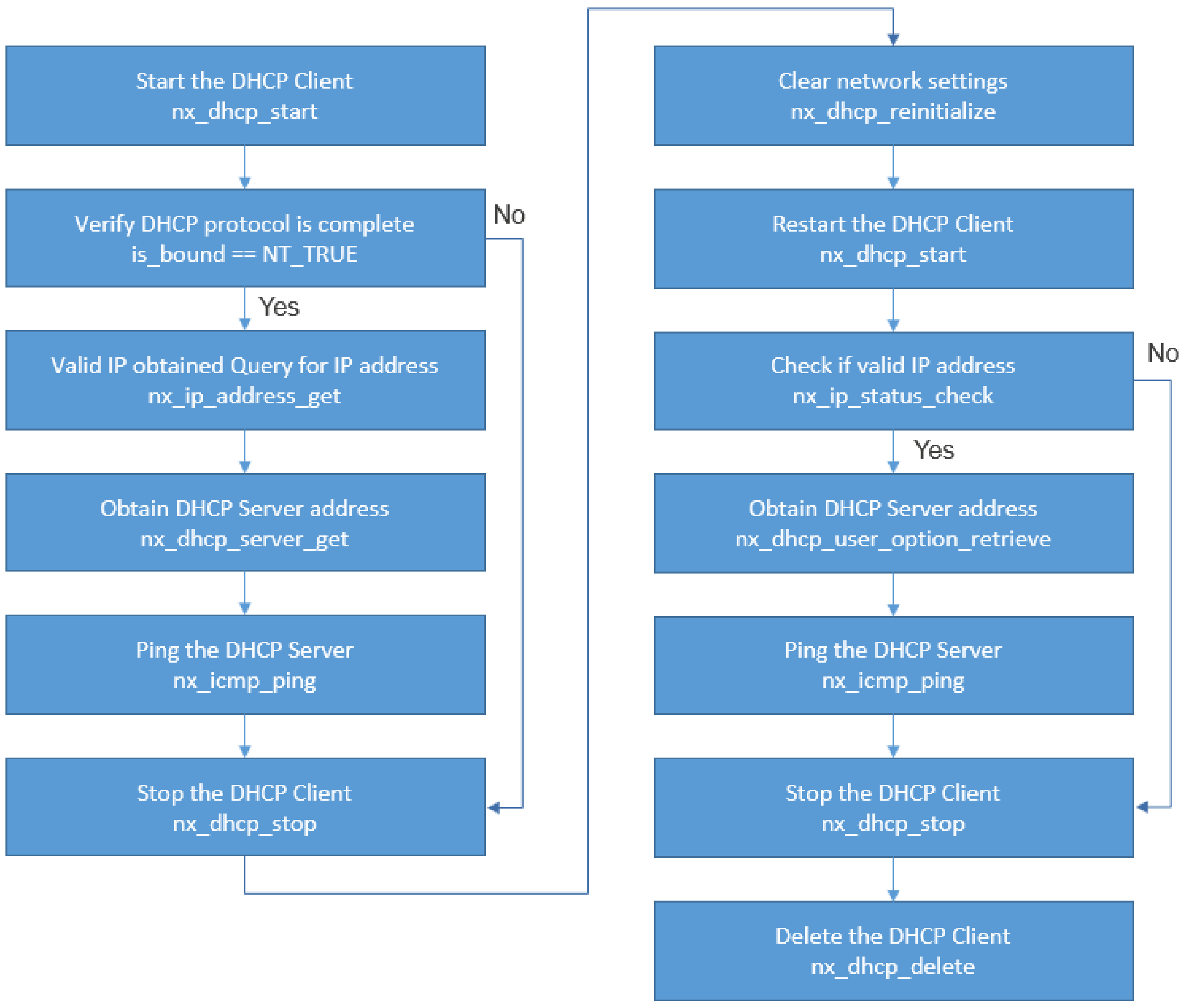

**Figure 7 NetX Duo DHCP IPv4 Client Module application project flow**

The `DHCPv4_Client_Netx_MG.c` file is located in the project once it has been imported into the ISDE. You can open this file within the ISDE and follow along with the following description to help identify key uses of APIs.

The progress of the DHCP protocol can also be viewed using a packet trace capture (such as Wireshark) running on any PC located on the same network; no special configuration is required since the DHCP packets in this application are broadcast and therefore visible to all hosts on the network.

At the top of dhcp_thread_entry(), the application waits a few seconds to let the IP thread task and the network driver get initialized. Both need to know a few things about each other before the NetX library calls work correctly. Note that we don't use the `nx_ip_status_check` service with the `NX_IP_ADDRESS_RESOLVED` because the application currently has a zero IP address. The thread-entry function calls the `run_dhcp_client_session` function to actually run the session.

**DHCP Client Properties**

The application project uses all the default settings for the NetX DHCP Client properties and has limited debug output (semi hosting.)

**Using the DHCP State Change Callback**

Before starting the DHCP Client, `run_dhcp_client_session` registers a DHCP state change callback function using the `nx_dhcp_state_change_notify` service. (This enables the application thread to be notified of DHCP state changes.) The advantage of this callback is the application thread need not wait or poll the IP instance for a valid IP address (e.g. the NX_DHCP_STATE_BOUND is achieved) and can perform other tasks instead. The callback function in this application only checks for the bound state.

**Starting the DHCP Client Thread Task**

The DHCP Client is started using the `nx_dhcp_start` service. This binds the client socket to the DHCP port and activates the client thread and timer. The application thread then periodically checks if the client is bound and relinquishes control to other threads to run before checking again. There is an upper limit to how long it waits to reach the bound state; this maximum wait is set by the WAIT_TO_BE_BOUND macro defined in DHCPv4_Client_Netx_MG.h as (2*NX_IP_PERIODIC_RATE). NX_IP_PERIODIC_RATE is used in NetX to convert timer ticks to seconds and automatically set internally in the Synergy environment.

**DHCP Services Available in the Bound State**

If the application is notified and the DHCP Client is bound before this wait time-out expires, `run_dhcp_client_session` queries the DHCP Client for the server's IP address using the `nx_dhcp_server_address_get` service. It gets its own IP address from the NetX service `nx_ip_address` set. At this point, the application can start sending and receiving non-broadcast packets. To demonstrate a valid IP address, `run_dhcp_client_session` pings the DHCP Server.

Once bound to an IP address, the DHCP Client thread task keeps track of the time remaining on the IP lease while it is running. When the lease is near to expiration, the DHCP Client thread initiates renewing the IP lease at the appropriate time automatically. In this application, you have to intentionally stop the DHCP Client so this does not happen.

**Stopping and Restarting the DHCP Client the First Time**

There may be instances where the application wants to stop the DHCP Client (for example, if it needs to be powered down or detects it is on another network.) The lease time assigned might be longer than the host needs, so there is no need to keep a thread running. In such an instance, before it can be restarted it must be stopped and reinitialized.

The application calls the `nx_dhcp_stop` service which suspends the DHCP Client thread task, unbinds the DHCP Client socket, and deactivates the DHCP Client timer. It also puts the DHCP Client in a state to be restarted. At this point, there is no DHCP Client thread activity. It does not clear network parameters so the application can still send and receive packets with the assigned IP address as the source.

To clear network parameters from NetX, (such as, IP address, network mask, and gateway) `run_dhcp_client_session` calls the `nx_dhcp_reinitialize` service. (This also clears the network parameters registered with the DHCP Client record.)

`run_dhcp_client_session` resets the is_bound flag; it then restarts the DHCP Client by calling the `nx_dhcp_start` service as before. This time, it calls `nx_ip_status_check` to check if it has received another IP address lease. Again, a maximum wait limit is set to wait for another IP address to be assigned. (There is no guarantee the server assigns the same IP address to the device.)

On reaching the bound state a second time, `run_dhcp_client_session` queries the DHCP Client for the DNS server using the `nx_dhcp_user_option_retrieve` service. This service can also be used to query the DHCP Client for IP lease time and other network information (see Section 3.1.2 for a list of options supported.)

To verify a valid IP address, `run_dhcp_client_session` pings the DNS Server.

**Stopping and Deleting the DHCP Client**

Before stopping the DHCP Client the second time, `run_dhcp_client_session` releases the IP address using the `nx_dhcp_release` service. This is not required but is considered 'good network behavior,' letting the DHCP server know the address is now available for other hosts. The DHCP Client can only release an IP address if it is bound to that IP address.

All DHCP Client activity is stopped with the `nx_dhcp_stop` call; the DHCP Client itself is then deleted (including the DHCP timer, threads, UDP socket) by calling the `nx_dhcp_delete` service.

To verify if any errors occurred, `run_dhcp_client_session` checks the error-counter variable for a non-zero value; this value is incremented each time an error occurs. If it is zero, `NX_SUCCESS` returns; otherwise, a non-zero error status is returned. The debug output using semi-hosting also records a trace of the DHCP session in the Renesas Debug Virtual Console.

The `DHCPv4_Client_Netx_MG.c` file also defines the DHCP state change callback function, `my_notify_callback`. This simply checks each notification for the state to be `NX_DHCP_STATE_BOUND` (bound state) and sets a flag for `run_dhcp_client_session` to check if an IP lease has been assigned. This flag must be cleared between stopping and restarting the DHCP Client thread as it is only a true/false value, not a counter.

Note: This description assumes you are familiar with using printf() the Debug Console in the Synergy Software Package. If you are unfamiliar with this, refer to the “How do I Use Printf() with the Debug Console in the Synergy Software Package” Knowledge Base article, available as described in the References section at the end of this document. Alternatively, the user can see results via the watch variables in the debug mode.

## DHCP IPv6 Client

The application project associated with this module guide demonstrates the aforementioned steps in a full design. The project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the NetX Duo DHCP IPv6 Client module. You can also read over the code (in DHCPv6_Client_Netxduo_MG.c) which is used to illustrate the NetX Duo DHCP IPv6 Client module APIs in a complete design.

This application project demonstrates the typical use of the NetX Duo DHCP IPv6 Client module APIs. It configures the DHCP Client with some common optional features before starting the DHCP Client thread task. It waits to be assigned a valid IP address and tests the IP address by pinging on the DHCP and DNS servers.

The `DHCPv6_Client_Netxduo_MG.c` file is located in the project once it has been imported into the ISDE. Note that this is not the auto generated code file `dhcpv6_thread0_entry.c`. The dhcpv6_thread0_entry calls run_dhcpv6_client_session, a function defined in the `DHCPv6_Client_Netxduo_MG.c` file to run a DHCP Client session. You can open `DHCPv6_Client_Netxduo_MG.c` within the ISDE and follow along with the following description to help identify key uses of APIs.

The following table identifies the target versions for the associated software and hardware used by the application project:

**Table 25 Software and Hardware Resources Used by the Application Project**

| Resource | Revision | Description |
|---|---|---|
| e$^2$ studio | 5.3.1 | Integrated Solution Development Environment |
| SSP | 1.2.0 | Synergy Software Platform |
| IAR EW for Renesas Synergy | 7.71.1 | IAR Embedded Workbench for Renesas Synergy |
| SSC | 5.3.1.002 | Synergy Standalone Configurator |
| SK-S7G2 | v3.0 to v3.1 | Starter Kit |

A NetX Duo Source element must be added to the project with the following values:

**Table 26 Configuration Settings for the NetX Duo Source**

| ISDE Property | Value | Description |
|---|---|---|
| NetX Duo IPv6 Support | Enabled | Enables IPv6 services in NetX Duo |
| Checksum computation support on transmitted ICMPv6 packets | Enabled | This should only be disabled if checksum support is handled in lower level driver/hardware. |
| Checksum computation support on received ICMPv6 packets | Enabled | This should only be disabled if checksum support is handled in lower level driver/hardware. |

The following figure shows a simple application project flow:

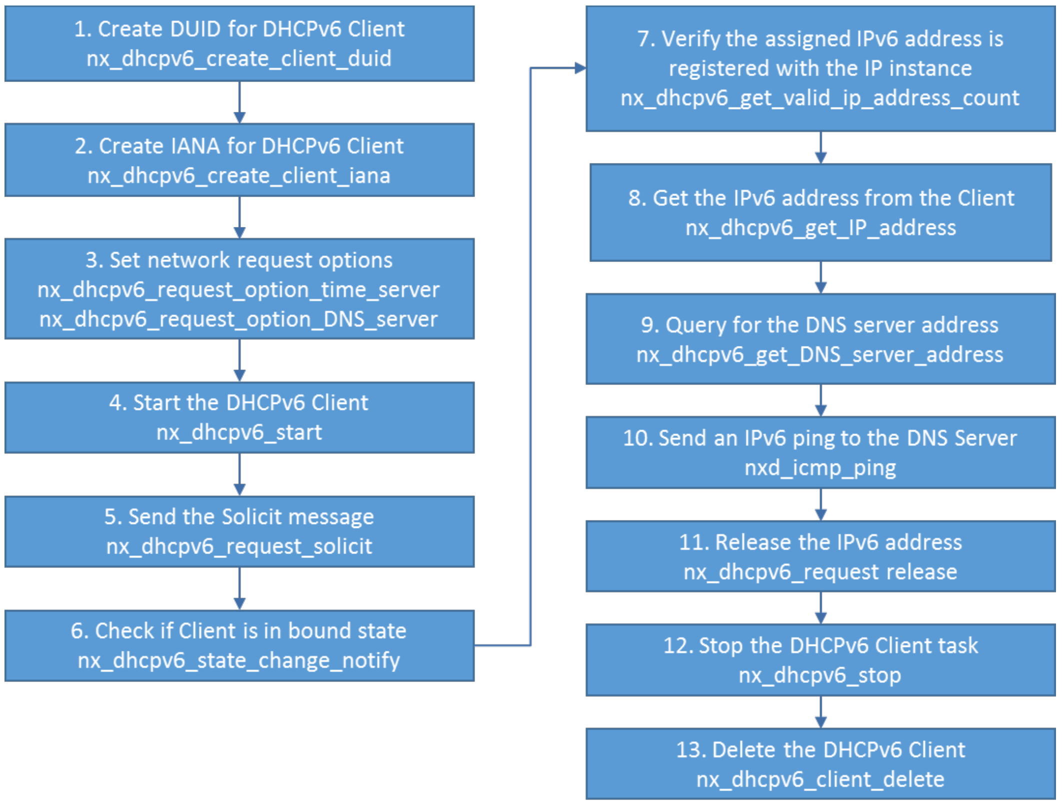
1. Create DUID for DHCPv6 Client
nx_dhcpv6_create_client_duid
2. Create IANA for DHCPv6 Client
nx_dhcpv6_create_client_iana
3. Set network request options
nx_dhcpv6_request_option_time_server
nx_dhcpv6_request_option_DNS_server
4. Start the DHCPv6 Client
nx_dhcpv6_start
5. Send the Solicit message
nx_dhcpv6_request_solicit
6. Check if Client is in bound state
nx_dhcpv6_state_change_notify
7. Verify the assigned IPv6 address is registered with the IP instance
nx_dhcpv6_get_valid_ip_address_count
8. Get the IPv6 address from the Client
nx_dhcpv6_get_IP_address
9. Query for the DNS server address
nx_dhcpv6_get_DNS_server_address
10. Send an IPv6 ping to the DNS Server
nxd_icmp_ping
11. Release the IPv6 address
nx_dhcpv6_request release
12. Stop the DHCPv6 Client task
nx_dhcpv6_stop
13. Delete the DHCPv6 Client
nx_dhcpv6_client_delete

**Figure 8 NetX Duo DHCP IPv6 Client Module application project flow**

The auto-generated file `dhcpv6_thread0_entry.c` calls `run_dhcpv6_client_session` to start the DHCPv6 protocol. The DHCPv6 Client is already created and uses its own packet pool (g_packet_pool1.) IPv6 and ICMPv6 are automatically enabled (DHCPv6 requires these protocols.)

The run_dhcpv6_client_session function sets up and start the DHCPv6 Client task for obtaining in IPv6 address.

In the properties of the g_dhcpv6_client0, there are two properties, *Name of state change notification function* and *Name of server error handler*. These default to `dhcpv6_state_change_notify` and `dhcpv6_server_error_handler`, respectively. Both these callbacks are defined in this application. The state change callback is limited to detecting the bound state, and sets a global flag when the DHCPv6 Client notifies it of the bound state. The server error handler is defined but has no functionality.

1. This function first creates a Client DUID (DHCP Unique Identifier, usually based on a MAC address) for the DHCPv6 Client using the `nx_dhcpv6_create_client_duid`. This is how the DHCPv6 Client identifies itself to the DHCPv6 server (who creates a 'server' DUID for the same purpose.)
2. Next, it creates an IANA for the DHCPv6 Client using the `nx_dhcpv6_create_client_iana`. This is the set of data that defines an IPv6 address. The server fills in this data block in the DHCPv6 message back to the client. The DHCPv6 Client extracts data from the IANA such as lease length and when to start the renewal process.
3. Most servers have additional network information they provide to the client. To request network options DNS server and time server, the function calls the `nx_dhcpv6_request_option_DNS_server` and `nx_dhcpv6_request_option_time_server` APIs. This adds the corresponding options into the DHCPv6 header option data.
4. Now it is ready to start the DHCPv6 Client and it calls the `nx_dhcpv6_start` API. This sets the DHCPv6 Client state and binds the client socket, in effect getting the DHCPv6 Client ready to exchange messages with the server.
5. To begin the DHCPv6 protocol, the application sends the SOLICIT message to the server using the `nx_dhcpv6_request_solicit` API. The DHCPv6 Client internally manages the rest of the DHCPv6 protocol.
6. The application checks a flag `is_bound` periodically to determine if the DHCPv6 Client is bound to an IPv6 address.
7. It is not necessary but a good idea to verify that the IPv6 address is registered with the IP instance. The application at this point calls the `nx_dhcpv6_get_valid_ip_address_count` API and verifies the address_count is not zero (this assumes the device had no previously existing global IPv6 addresses.) Because Duplicate Address

Detection is enabled, the application waits a few seconds for it to complete. The DHCPv6 Client IPv6 address is now marked valid and ready for use.

8. While the application has no specific requirement to know its IPv6 address, it calls the `nx_dhcpv6_get_IP_address` API and displays the result.
9. To test the IPv6 address, the application queries the DHCPv6 Client for the DNS server address by calling the `nx_dhcpv6_get_DNS_server_address` API.
10. Now the application has a host it can send a ping to. It does so by calling nxd_icmp_ping to the IPv6 DNS server address. This NetX Duo service can also accept IPv4 packets in the NXD_ADDRESS data instance by setting the `nxd_ip_version` field to NX_IP_VERSION _V4.
11. If the DHCPv6 task was left running, it would keep track of the time remaining on the lease and begin the renewal process as instructed by the server (using the T1 and T2 times.) However, in this application we stop the DHCPv6 Client. To comply with good network practices, we release the IPv6 address back to the server by calling the `nx_dhcpv6_request_release` API.
12. To stop the DHCPv6 Client thread, the application calls `nx_dhcpv6_stop`. This also unbinds the socket port and stops the client thread and timers.
13. The last step is to delete the DHCPv6 Client itself; this is done by calling `nx_dhcpv6_client_delete`.

Note: The packet pool that was supplied in the creation of the DHCPv6 Client is still in use. In a memory constrained system, it might be necessary to release the packet pool memory. The application can call `nx_packet_pool_delete`() to remove that packet pool if it is not used by another task. In this application project, it is shared with the IP instance so we do not delete it.

The application project description assumes you are familiar with using printf() with the Debug Console in the Synergy Software Package. If you are unfamiliar with printf(), see *How do I Use Printf() with the Debug Console in the Synergy Software Package* Knowledge Base article, listed in the References section. Alternatively, the user can see results via the watch variables in the debug mode.

## 8. Customizing the NetX Duo DHCP Client Module for a Target Application

A brief description of DHCP Client configuration settings commonly changed in a DHCP Client applications follows; the details are given in the context of this application project:

### DHCP IPv4 Client

#### Requesting a Specific IP address

The application may wish to request a specific IP address from the DHCP server. If so, it should call the `nx_dhcp_request_client_ip` API before `nx_dhcp_start` is called in this application project. This is often useful for a device that wants to keep the same IP address. Also, by setting the third input in this API, `skip_discover_message`, to NX_TRUE, the application is requesting that the DISCOVER and OFFER steps be omitted (a normal transaction is DISCOVER -> OFFER -> REQUEST -> ACK.) This reduces the DHCP protocol to one REQUEST message from the client and one ACK message from the server. It is up to the server to accommodate either or both requests; in the event it does not, the DHCP Client automatically reverts to the normal four-step process.

#### Supplying a user-defined packet pool for the DHCP Client

The application may wish to supply the packet pool to the DHCP Client rather than the client creating its packet pool which it is defaulted to do. The advantages of using separate packet pools for different components of an application is that memory can be optimized for the expected usage; tt also simplifies tracking down packet pool leaks and/or problems with packet pool-depletion.

To do so, enable the use application packet pool property of the DHCP Client stack element. Then create a separate packet pool from the one used by the IP instance (usually designated g_packet_pool0). Click on the + icon in the Thread Stack elements and choose X-Ware -> NetX -> Netx Packet Pool Instance. This creates a packet pool for the DHCP Client. Note that the packet pool payload must be a minimum of 576 bytes plus the size of the Ethernet frame header.

Before calling the `nx_dhcp_start` API, register the new packet pool (which defaults to g_packet_pool1) with the DHCP Client by calling the `nx_dhcp_packet_pool_set` API. It is up to the application to delete the packet pool when it no longer has use for it. The DHCP Client deletes the packet pool only if it created it in the first place.

#### Enabling ARP Probes

The RFC 2131 recommends that a device using DHCP to obtain an IP address verify that it is not in use by another host; this is done by sending out ARP probes immediately after being assigned the IP address. If no hosts respond to these probes, the device can assume the address is not in use. The NetX DHCP Client performs this service if the "sending ARP probes" property is enabled on the DHCP Client stack element. The DHCP Client is set up with a callback which NetX uses to notify the DHCP Client of an ARP conflict; in this manner, the DHCP Client can verify the uniqueness of an IP address before promoting the DHCP Client to the bound state. If another host does respond, then the DHCP Client sends a DECLINE message to the server and automatically restarts the DHCP protocol at the INIT state.

If "sending ARP probes" is enabled, the application must expect to wait longer to achieve the bound state. The number of probes is specified by the ARP probe count property and defaults to 2. The interval to wait for a response (and before sending the next probe) is specified by the ARP probe wait time which defaults to 1 second. There is also a random delay imposed on the wait time. The minimum and maximum wait time is specified by maximum ARP probe wait time (seconds) and maximum ARP probe wait time (seconds) properties which default to 1 and 2 seconds, respectively. Using these defaults, the WAIT_TO_BE_BOUND defined in DHCPv4_Client_Netx_MG.h should be adjusted to (4*NX_IP_PERIODIC_RATE) to allow enough time for the ARP Probe process to complete and the DHCP Client to achieve the bound state.

#### Accessing DHCP Servers behind a Router

NetX DHCP Client has a clear broadcast flag service which is intended to allow a device to access a server behind a router outside the local network; this can only happen if the DHCP Server sends unicast packets through the router to the client. Because the device has no IP address, it would normally be unable to receive a unicast OFFER or ACK from the server; however, the NetX IP layer does accept packets with a zero destination IP address. The packet is forwarded up to the UDP layer and onto the DHCP Client itself. The DHCP Client can signal the server to send unicast packets by calling `nx_dhcp_clear_broadcast_flag` service with the clear_flag input set to NX_TRUE.

#### More Uses of the State Change Callback

In the event the application fails to achieve the bound state, the callback function in this application can be expanded to check other states the DHCP Client has achieved, if not the bound state. This may help resolve the reason why a DHCP session failed to progress to the bound state. For example, if the DHCP Client does not reach at least the NX_DHCP_STATE_REQUESTING, it has not received any Server responses. If it reached the NX_DHCP_STATE_ARP_PROBING, and then is reset to the NX_DHCP_STATE_INIT again, that indicates the IP address assigned was not unique and the Client has started the protocol over again.

Note: NX_DHCP_STATE_INIT is set each time the application calls nx_dhcp_start. If this state is reached again it indicates a failure in one of the states that leads to the bound state and the DHCP Client automatically resets it to the NX_DHCP_STATE_INIT state.

### DHCP IPv6 Client

#### DHCPv6 Client Callbacks

The application project does not make any use of the server error-handler. An application may find it useful to be informed of server error messages (such as unable to assign IPv6 address.). Otherwise, all DHCPv6 protocol is handled internally and the application has no other way of knowing if there is a problem with the server.

Similarly, the state change callback can be expanded to check for a return to the initial (INIT) state or when the lease is close to renewal time. That might affect operations that use NetX Duo services and the current IPv6 address.

# 9. Running the NetX Duo DHCP Client Module Application Project

To run the NetX Duo DHCP Client module application project and to see it executed on a target kit, you can simply import it into your ISDE, compile, and run debug. See the included application note, *Importing a Renesas Synergy Project*, for instructions on importing, building, and running a project in e$^2$ studio ISDE or IAR EW for Synergy.

To implement the NetX Duo DHCP Client module application in a new project, follow the steps for defining, configuring, auto-generating files, adding code, compiling, and debugging on the target kit. Following these steps is a hands-on approach that helps make the development process with SSP more practical, while just reading over this guide tends to be more theoretical.

Note: The following steps are described in sufficient detail for someone experienced with the basic flow through the Synergy development process. If these steps are not familiar, refer to the first few chapters of the SSP User's Manual for a description of how to accomplish these steps.

### DHCP IPv4 Client

To create and run the DHCP IPv4 Client application project, simply follow these steps:

1. Create a new Renesas Synergy project for the S7G2-SK called DHCP IPv4_Client_App.
2. Select the **Threads** tab. Add a thread for the DHCP Client application and set the Name to "**dhcp_thread**" (to generate the `dhcp_thread_entry.c` file).
3. Set the Stack size to 2048 and the Priority to 3 (the IP thread task should run at the highest priority, and is set to 1.)
4. In the thread stack pane, click on the + icon and choose **X-Ware** -> **NetX Duo** -> **Protocols** -> **NetX Duo DHCP IPv4 Client**.
5. Use the default properties for the DHCP Client instance.
6. For the NetX Port ETHER driver, make sure to set the Ethernet interrupt priority; its default value is Disabled. For the SK-S7G2 board, choose Channel 1 and the Channel 1 Phy Reset Pin should be IOPORT_PORT_08_PIN_06.
7. Click the **Add NetX Duo Packet Pool** and choose either **Use** (the `existing g_packet_pool0`) or **New** (create `g_packet_pool1`.)
8. Click the "**Generate Project Content**" button.
9. Add the code from the supplied project file "**DHCPv4_Client_Netxduo_MG.c**" into the generated "**dhcp_thread_entry.c**" file so that the latter can call run_dhcp_client_session(). Similarly, move the settings in the `DHCPv4_Client_Netxduo_MG.h` file to the top of `dhcp_thread_entry.`c.
10. As an alternative to Step 8, right click on the project and choose **New** -> **Source File**. Enter the name of the new source file, such as `DHCP IPv4_client_app.c`. Copy the code from the supplied project file "**DHCPv4_Client_Netxduo_MG.c**" file into this file.
11. Right click on the project and choose **New** -> **Header File**. Enter the name of the new header file, such as **DHCPv4_client_app.h**. Copy the code from the supplied project file "**DHCPv4_Client_Netxduo_MG.h**" file into this file.
12. For optimal debugging, set optimization to none -O0. For improved performance, set it to -O2.
13. There are two ways to enable/disable debug output: right click the project and choose **Properties** -> **C/C++ Build** -> **Settings** -> **Optimization**. Set the Optimization Level to None (-O0) or (-O2).
    - The header file includes a #define for SEMI_HOSTING. If this is defined, debug output is enabled.
    - Right click the project and choose **Properties** -> **C/C++ Build** -> **Settings** -> **Cross ARM C Compiler** -> **Preprocessor** and add a define in the Defined symbols pane by clicking on the + icon. Enter **SEMI_HOSTING**.
14. Right click the project and choose **Build Project**.
15. Connect to the host PC via a micro USB cable to J19 on SK-S7G2. Connect an Ethernet cable to the RJ45 connector on J11.
16. Right click the project and choose Debug as -> **Renesas GDB Hardware Debugging**.
17. Run the application.
18. The output can be viewed in the Renesas Debug Console:

```
DHCP Client is assigned an IP address lease.
DHCP Client address is 0xc00202d2.
DHCP Server IP address is 0c0020201.
Successfully pinged Server.
DHCP Client is reinitializing...
Starting debug output...DHCP Client is assigned IP address lease for a second time.
Successfully pinged Server.
Released IP address back to Server.
Stopping the DHCP Client.
DHCP Client demo has completed with 0 errors.
```

**Figure 9 Example Output from DHCP IPv4 Client Application Project**

### DHCP IPv6 Client

To create and run the DHCP IPv6 Client application project, simply follow these steps:

1. Create a new Renesas Synergy project for the S7G2-SK called DHCP IPv6_Client_App.
2. Select the **Threads** tab. Add a thread for the DHCPv6 Client application and set the Name to "**dhcpv6_thread0**" (to generate the `dhcpv6_thread0_entry.c` file.)
3. Set the Stack size to 2048 and the Priority to 3 (the IP thread task should run at the highest priority, and is set to 1.)

4. In the thread stack pane, click the + icon and choose **X-Ware** -> **NetX Duo** -> **Protocols** -> **NetX Duo DHCP IPv6 Client**.
5. Use the default properties for the DHCP IPv6 Client instance
6. If you wish to add the NetX Duo Source element, set the *Checksum computation support on received ICMPV6 packets* and *Checksum computation support on transmitted ICMPv6 packets* to enabled. Also, check the *NetX Duo IPv6 Support* is also enabled. (These properties are automatically enabled when the NetX Duo Source element is not added to the project.)
7. For the NetX Port ETHER driver, make sure to set the Ethernet interrupt priority; the default value is **disabled**. For the SK-S7G2 board, choose Channel 1 and the Channel 1 Phy Reset Pin should be IOPORT_PORT_08_PIN_06.
8. Click the "**Generate Project Content**" button.
9. Add the code from the supplied project file **DHCPv6_Client_Netxduo_MG.c** into the generated "**dhcpv6_thread0_entry.c**" file so that the latter can call `run_dhcpv6_client_session`. Similarly, move the settings in the `DHCPv6_Client_Netxduo_MG.h` file to the top of `dhcpv6_thread0_entry.c`.
10. As an alternative to Step 8, right click the project and choose **New** -> **Source File**. Enter the name of the new source file, such as `DHCPv6_client_app.c`. Copy the code from the supplied project file "`DHCPv6_Client_Netxduo_MG.c`" file into this file.
11. Right click the project and choose **New** -> **Header File**. Enter the name of the new header file, such as `DHCPv6_client_app.h`. Copy the code from the supplied project file "`DHCPv6_Client_Netxduo_MG.h`" file into this file.
12. For optimal debugging, set optimization to none -O0. For improved performance, set it to -O2 (Optimize more)
13. There are two ways to enable/disable debug output: right click the project and choose **Properties** -> **C/C++ Build** -> **Settings** -> **Optimization**. Set the Optimization Level to None (-O0) or Optimize more (-O2).
14. The `DHCPv6_Client_Netxduo_MG.h` header file includes a #define for SEMI_HOSTING. This enables debug output.
15. As an alternative to Step 11, right click the project and choose **Properties** -> **C/C++ Build** -> **Settings** -> **Cross ARM C Compiler** -> **Preprocessor** and add a define in the Defined symbols pane by clicking on the + icon. Enter **SEMI_HOSTING**.
16. Right click the project and choose **Build Project**.
17. Connect to the host PC via a micro USB cable to J19 on SK-S7G2. Connect an Ethernet cable to the RJ45 connector on J11.
18. Right click the project and choose Debug as -> **Renesas GDB Hardware Debugging**.
19. Run the application.
20. The output can be viewed in the Renesas Debug Console:

```
The DHCPv6 Client is started
Sent the Solicit message. Waiting for bound state...
Client is bound to IPv6 address 0x20010db8 0x0 0x0 0x73
DNS Server address is  0x20010db8 0x0 0x0 0x89
Pinged the DNS Server!
Stopping and deleting the DHCPv6 Client
DHCPv6 Client completed with 0 errors!
```

## 10. NetX Duo DHCP Client Module Conclusion

This module guide provides the background information needed to select, add, configure, and use the module in an example project. Many of these steps were time consuming and error-prone activities in previous generations of embedded systems.

For example, preparing NetX Duo with the required services, enabling UDP and establishing an IP instance, and defining the driver interface for NetX Duo. Access to the DHCP Client element properties box simplifies customizing the DHCP Client properties, rather than needing to track down the configuration options in the source code header files and manually setting their value.

The Renesas Synergy Platform makes these tasks less time consuming and removes common errors, like conflicting configuration settings or incorrect selection of lower-level drivers. The use of high-level APIs showcased in the application project highlights development time savings by allowing work to begin at a high level and avoiding the time required in older development environments to use or, in some cases, create, lower-level drivers.

At the same time, the application project shows the use of calling DHCP Client services directly. At some point, an application may need to write the entire project directly from source code to fully customize the project to the application. That process becomes much easier after working through this application project and gaining some familiarity with the NetX Duo DHCP Client.

## 11. NetX Duo DHCP Client Module Next Steps

After you have mastered a simple DHCP Client module project, you may want to review more complex examples.

### DHCP IPv4 Client

#### Suspending and Restoring the DHCP Client State

If your application requires the ability to restore the DHCP Client after the device is powered off, you would need to use the DHCP Client Restore State feature. DHCP Clients with short IP leases in particular would need to save and restore the DHCP Client record to insure timely renewal of an IP address. This is a good example of combining the convenience of developing on the Synergy Platform with customized coding by the developer to maximize DHCP Client capabilities.

The developer sets this feature by enabling the *Persistent client state* property. Before powering off, the client record must be extracted and saved to non-volatile memory. A client record is created using this service:

```
UINT _nx_dhcp_client_create_record(NX_DHCP *dhcp_ptr)
```

On powering up, the saved client record is restored by calling this service from the application thread:

```
UINT _nx_dhcp_client_restore_record(NX_DHCP *dhcp_ptr, NX_DHCP_CLIENT_RECORD
*client_record_ptr, ULONG time_elapsed)
```

In some usage cases, the DHCP Client thread may only need to be suspended and resumed. To keep the time remaining on the client lease up to date, the client lease-time remaining needs to be updated for the time interval over which the DHCP Client thread task is suspended. Doing so also requires *Persistent client state* to be enabled; in this case, however, no record needs to be extracted from the DHCP Client (although the time remaining on the lease needs to be updated.)

The services for suspending and resuming the DHCP Client thread are:

```
UINT _nx_dhcp_suspend(NX_DHCP *dhcp_ptr)
```

```
UINT _nx_dhcp_resume(NX_DHCP *dhcp_ptr)
```

The service for updating the DHCP Client lease-time remaining is:

```
UINT _nx_dhcp_client_update_time_remaining(NX_DHCP *dhcp_ptr, ULONG time_elapsed)
```

Contact Renesas Synergy support for more detailed information on managing the DHCP Client around power cycles and suspension of the DHCP Client.

#### Running the DHCP IPv4 Client on a Secondary Interface

To support a secondary interface, add a NetX Duo Source element to the project and set the *Maximum Physical Interfaces* property to 2 (if only one secondary interface exists.) The application must then register the interface with the NetX Duo IP instance by calling the `nx_ip_interface_attach` service. (The board must have two network Ethernet connections.)

Set the interface index by calling the `nx_dhcp_set_interface_index` service before starting the DHCP Client. The NetX Duo DHCP Client defaults to the primary interface (index = 0.) The `nx_dhcp_set_interface_index` service sets the DHCP Client to the network interface specified by the index input. In the case of one secondary interface, this would be 1.

After the DHCP Client and NetX Duo are set up for the secondary interface, the DHCP Client session is run no differently from a DHCP Client on a primary interface.

Contact Renesas Synergy support for more detailed information on setting up a secondary interface.

### DHCP IPv6 Client

#### Suspending and Restoring the DHCP IPv6 Client State

The DHCPv6 Client protocol requires that the DHCPV6 Client implementation have a means to save and restore the client state using non-volatile memory. This, as with the DHCP IPv4 Client, allows a device that has been power-cycled to restore and update the client state. Similarly, if an application suspends the DHCPv6 Client, and then resumes it, it can update the time remaining on the lease if it knows the length of time the client was suspended. The DHCPv6 Client services for this task update the time remaining on the lease with that data and determine if the DHCPv6 state changed while the client task was suspended or the device was powered off. DHCPv6 Clients with short IP leases in particular would need to save and restore the DHCP Client record to insure timely renewal of an IP address.

To enable this feature, define the symbol NX_DHCPV6_CLIENT_RESTORE_STATE. The easiest way to do this is to right click the project -> Properties -> C/C++ Build -> Settings -> Cross Arm C Compiler -> Preprocessor and type in the symbol.

Note this feature is only applicable to the DHCPv6 Client bound to an IPv6 address.

These are the API that are used to stop a DHCPv6 Client and start it again:

- `nx_dhcpv6_stop`(NX_DHCPV6 *dhcpv6_ptr);
- `nx_dhcpv6_start`(NX_DHCPV6 *dhcpv6_ptr);

This API creates a record of the current client state after stopping the client task:

```
nx_dhcpv6_client_get_record((NX_DHCPV6 *dhcpv6_ptr,
                      NX_DHCPV6_CLIENT_RECORD *client_record_ptr);
```

This API updates the current DHCPv6 Client instance with the saved record and applies the length of time the client was stopped to the time remaining on the lease with the time_elapsed input:

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

The application then starts the DHPCv6 Client using the `nx_dhcpv6_start` API. There is no need to call `nx_dhcpv6_request_solicit` because the DHCPv6 Client state is in the bound state.

Contact Renesas Synergy support for more detailed information on managing the DHCP Client around power cycles and suspension of the DHCP Client.

**Running the DHCP IPv6 Client n a Secondary Interface**

To run a DHCP IPv6 Client on a secondary interface (similarly to the DHCP IPv4 Client), set the *Maximum Physical Interfaces* property of the NetX Duo Source stack element to 2 (if only one secondary interface exists.). The application must then register the interface with the NetX Duo IP instance by calling the `nx_ip_interface_attach` service. (The board must have two network Ethernet connections.)

Set the interface index by calling the `nx_dhcpv6_set_interface_index service` before starting the DHCP IPv6 Client. The NetX Duo DHCP IPv6 Client defaults to the primary interface (index = 0). The `nx_dhcpv6_set_interface_index` service sets the DHCP IPv6 Client to the network interface specified by the index input. In the case of one secondary interface, this would be 1.

After the DHCP IPv6 Client and NetX Duo are set up for the secondary interface, the DHCPv6 Client session is run no differently from a DHCPv6 Client on the primary interface.

Contact Renesas Synergy support for more detailed information on setting up a secondary interface.

# 12. NetX Duo DHCP Client Module Reference Information

SSP User Manual: Available in html format in the SSP distribution package and also as pdf from the Synergy Gallery.

To find the most up to date reference materials and their locations, visit the Synergy Knowledge Base and do a search for the module name and include “module guide references” in the search.

### DHCP IPv4 Client

If you are looking for the References for the DHCP (IPv4) Client module, visit https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy%E2%84%A2_Platform/Renesas_Synergy_Knowledge_Base and enter “**dhcp module guide references**” in the search bar. The search brings up a list of results, and the top one becomes the References Page for that Module Guide. The following URL takes you directly to the search results for the example.

https://en-us.knowledgebase.renesas.com/Special:Search?fpid=230&search=dhcp%20client%20module%20guide&path=&limit=55&page=1&q=dhcp%20client%20module%20guide%20references&tags=

### DHCP IPv6 Client

If you are looking for the References for the DHCP IPv6 Client module, visit https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy%E2%84%A2_Platform/Renesas_Synergy_Knowledge_Base and enter “**dhcpv6 module guide references**” in the search bar. The search brings up a list of results, and the top one becomes the References Page for that Module Guide. The following URL takes you directly to the search results for the example.

https://en-us.knowledgebase.renesas.com/Special:Search?fpid=230&search=dhcpv6%20module%20guide&path=&limit=55&page=1&q=dhcp%20client%20module%20guide%20references&tags=

## Website and Support

Support: https://synergygallery.renesas.com/support

Technical Contact Details:

- America: https://www.renesas.com/en-us/support/contact.html
- Europe: https://www.renesas.com/en-eu/support/contact.html
- Japan: https://www.renesas.com/ja-jp/support/contact.html

## Revision History

| | | Description | |
|---|---|---|---|
| **Rev.** | **Date** | **Page** | **Summary** |
| 1.00 | 1/9/2018 | | Initial Release |

RENESAS

**SALES OFFICES** Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

**Renesas Electronics America Inc.**
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

**Renesas Electronics Canada Limited**
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

**Renesas Electronics Europe Limited**
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

**Renesas Electronics Europe GmbH**
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

**Renesas Electronics (China) Co., Ltd.**
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

**Renesas Electronics (Shanghai) Co., Ltd.**
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

**Renesas Electronics Hong Kong Limited**
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

**Renesas Electronics Taiwan Co., Ltd.**
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

**Renesas Electronics Singapore Pte. Ltd.**
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

**Renesas Electronics Malaysia Sdn.Bhd.**
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

**Renesas Electronics India Pvt. Ltd.**
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

**Renesas Electronics Korea Co., Ltd.**
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0