Renesas Synergy™ Platform

NetX™ and NetX Duo™ DNS Client Module Guide

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.

Domain Name System (DNS) is a distributed database containing the mapping between domain names and physical IP addresses. The database is referred to as “distributed” because no single entity on the Internet contains the complete mapping. An entity, called a DNS server, maintains a portion of the mapping. Numerous DNS servers exist on the Internet, each of which contains a subset of the database. DNS servers also respond to DNS client requests for domain name mapping information, only if the server has the requested mapping. The DNS client protocol for NetX and NetX Duo provides the application with services to request mapping information from one or more DNS Servers.

Note: Except where noted, the NetX Duo DNS Client is identical to the NetX DNS Client in application set-up and running of DNS queries. To set up the IP instance for IPv6 in NetX Duo, see the *NetX Duo User Guide* for the Renesas Synergy™ Platform.

Key elements related to the NetX and NetX Duo DNS Client module implementation on the Renesas Synergy Platform are provided. The primary focus is to add and configure the NetX and NetX Duo DNS Client Module to a Renesas Synergy Platform project. For details, see *NetX™ Domain Name System (DNS) Client User Guide* for the Renesas Synergy™ Platform or *NetX Duo™ Domain Name System (DNS) Client User Guide* for the Renesas Synergy™ Platform. These documents reside in the *X-Ware™ Component Documents for Renesas Synergy™* zip available for download at the Renesas Synergy Gallery site (www.renesas.com/synergy/software).

Contents

1. NetX and NetX Duo DNS Client Module Features 2
2. NetX and NetX Duo DNS Client Module APIs Overview 3
3. NetX and NetX Duo DNS Client Module Operational Overview 5
3.1 NetX and NetX Duo DNS Client Module Important Operational Notes and Limitations 8
3.1.1 NetX and NetX Duo DNS Client Module Operational Notes 8
3.1.2 NetX and NetX Duo DNS Client Module Limitations 9
4. Including the NetX and NetX Duo DNS Client Module in an Application 9
5. Configuring the NetX and NetX Duo DNS Client Module 10
5.1 Configuration Settings for the NetX DNS Client Lower-Level Modules 11
5.2 NetX and NetX Duo DNS Client Module Clock Configuration 14
5.3 NetX and NetX Duo DNS Client Module Pin Configuration 14
6. Using the NetX and NetX Duo DNS Client Module in an Application 14
7. The NetX and NetX Duo DNS Client Module Application Project 15
8. Customizing the NetX and NetX Duo DNS Client Module for a Target Application 17
9. Running the NetX and NetX Duo DNS Client Module Application Project 18

10. NetX and NetX Duo DNS Client Module Conclusion 20
11. NetX and NetX Duo DNS Client Module Next Steps 20
12. NetX and NetX Duo DNS Client Module Reference Information 20
Revision History 22

1. NetX and NetX Duo DNS Client Module Features

- Optional creation of separate packet pool for DNS operations
- Support of Type A, AAAA, and NS DNS queries
- Support of CNAME, SRV, TXT, SOA, and MX DNS resource types
- Support for DNS cache for storing and retrieving cached DNS data
- High-level APIs for
  - Name and IP address lookup
  - Adding and removing DNS servers
  - Creating and deleting the DNS instance
- NetX DNS is compliant with the following RFCs:
  - RFC1034 Domain Names – Concepts And Facilities
  - RFC1035 Domain Names — Implementation And Specification
  - RFC1480 The US Domain
  - RFC 2782 A DNS RR for specifying the location of services (DNS SRV)
  - RFC 3596 DNS Extensions to Support IP Version 6

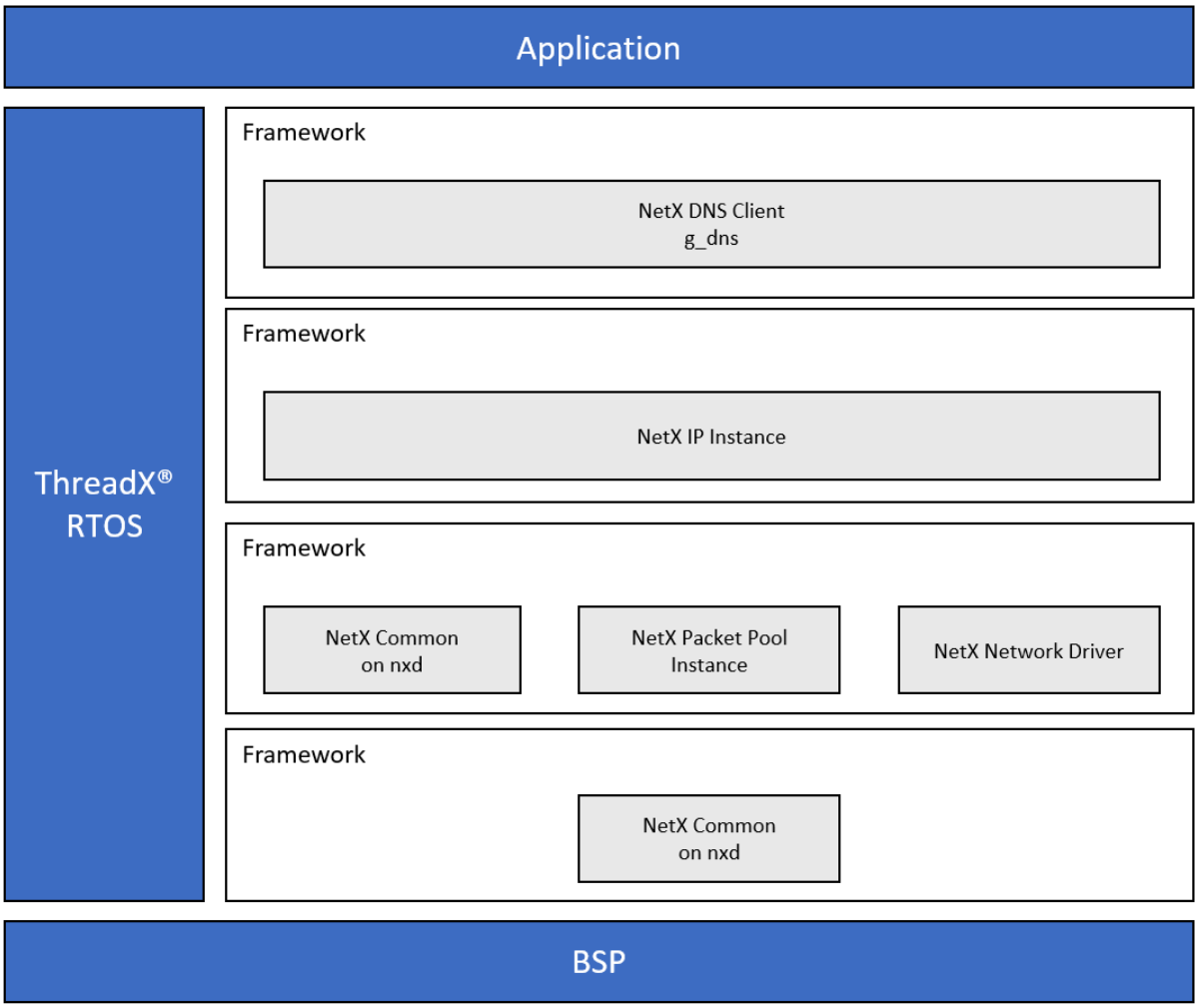


Figure 1. NetX and NetX Duo DNS Client Module

2. NetX and NetX Duo DNS Client Module APIs Overview

The NetX and NetX Duo DNS Client module defines APIs for connecting, binding, listening, sending, and receiving. A complete list of the available APIs, an example API call, and a short description of each API can be found in the following table. The end of the summary table lists *nx* services unique to NetX and NetX Duo BSD. Following the API summary table is a list of status return values.

Table 1. NetX and NetX Duo DNS Client Module API Summary

Function Name	Example API Call and Description
nx_dns_create	`nx_dns_create(&my_dns, &my_ip, "My DNS");` Create a DNS Client instance.
nx_dns_delete	`nx_dns_delete(&my_dns);` Delete a DNS Client instance.
nx_dns_packet_pool_set	`nx_dns_packet_pool_set(&my_dns, &packet_pool);` Set the DNS Client packet pool.
nx_dns_get_serverlist_size	`nx_dns_get_serverlist_size (&my_dns, 5);` Return the size of the DNS Client's Server list.
nx_dns_info_by_name_get	`nx_dns_info_by_name_get(&my_dns, "www.abc1234.com", &ip_address, &port,` 200); Return IPv4 address, port querying on input host name.

nx_dns_ipv4_address_by_name_get	`nx_dns_ipv4_address_by_name_get(&client_dns,(UCHAR *)"www.my_example.com",`
	`record_buffer,sizeof(record_buffer),&record_count,500);`
	Look up the IPv4 address for the specified host name.

nx_dns_host_by_address_get	`nx_dns_host_by_address_get(&my_dns, IP_ADDRESS(192.2.2.10), &resolved_name[0], BUFFER_SIZE, 450);` Look up a host name from a specified IP address (supports only IPv4 addresses).
nx_dns_host_by_name_get	`nx_dns_host_by_name_get(&my_dns, "www.my_example.com", &ip_address, 4000);` Look up an IP address from the host name (supports only IPv4 addresses).
nx_dns_server_add	`nxd_dns_server_add(&my_dns, server_address);` Add a DNS Server of the specified IP address to the Client's Server list (supports only IPv4).
nx_dns_server_get	`nx_dns_server_get(&my_dns, 5, &my_server_address);` Return the DNS Server in the Client list at the specified index (supports only IPv4 addresses).
nx_dns_server_remove	`nx_dns_server_remove(&my_dns, IP_ADDRESS(202,2,2,13));` Remove a DNS Server from the Client list (supports only IPv4 addresses).
nx_dns_server_remove_all	`nx_dns_server_remove_all(&my_dns);` Remove all DNS Servers from the Client list.
nx_dns_cache_initialize	`nx_dns_cache_initialize(&my_dns, &dns_cache, 2048);` Initialize a DNS Cache.
nx_dns_cache_notify_clear	`nx_dns_cache_notify_clear(&my_dns);` Clear the DNS cache full notify function.
nx_dns_cache_notify_set	`nx_dns_cache_notify_set(&my_dns, cache_full_notify_cb);` Set the DNS cache full notify function.

nx_dns_cname_get	`nx_dns_cname_get(&client_dns, (UCHAR *)"www.my_example.com",`
	`record_buffer, sizeof(record_buffer), 500);`
	Look up the canonical domain name for the input domain name alias.

Function Name	Example API Call and Description
nx_dns_authority_zone_start_get	nx_dns_authority_zone_start_get(&client_dns, (UCHAR *)"www.my_example.com", record _buffer, sizeof(record_buffer), &record_count, 500); Look up the start of a zone of authority associated with the specified host name.
nx_dns_domain_name_server_get	nx_dns_domain_name_server_get(&client_dns, (UCHAR *)" www.my_example.com ", record_buffer, sizeof(record_buffer), &record_count, 500); Look up the authoritative name servers for the input domain zone
nx_dns_domain_mail_exchange_get	nx_dns_domain_mail_exchange_get(&client_dns, (UCHAR *)" www.my_example.com ",record_buffer, sizeof(record_buffer), &record_count, 500); Look up the mail exchange associated with the specified host name.
nx_dns_domain_service_get	nx_dns_domain_service_get(&client_dns, (UCHAR *)“www.my_example.com ", record_buffer, sizeof(record_buffer), &record_count, 500); Look up the service(s) associated with the specified host name.
nxd_dns_ipv6_address_by_name_get**	nxd_dns_ipv6_address_by_name_get(&client_dns, (UCHAR *)"www.my_example.com", record__buffer, sizeof(record_buffer), &record_count, 500); Look up the IPv6 address from the specified host name.
nxd_dns_host_by_address_get**	nxd_dns_host_by_address_get(&my_dns, &host_address, resolved_name, sizeof(resolved_name), 4000); Look up a host name from the input IP address (supports both IPv4 and IPv6 addresses).
nxd_dns_host_by_name_get**	nxd_dns_host_by_name_get(&my_dns, “www.my_example.com”, &ip_address, 4000, NX_IP_VERSION_V4); Look up an IP address from the input host name (supports both IPv4 and IPv6 addresses).
nxd_dns_server_add**	nxd_dns_server_add(&my_dns, &server_address); Add the input DNS Server to the Client server list (supports both IPv4 and IPv6 addresses).
nxd_dns_server_get**	nxd_dns_server_get(&my_dns, 5, &my_server_address); Return the DNS Server in the Client list at the specified index (supports both IPv4 and IPv6 addresses).
nxd_dns_server_remove**	nxd_dns_server_remove(&my_dns,& server_ADDRESS); Remove a DNS Server of the specified IP address from the Client list (supports both IPv4 and IPv6 addresses).

Note: For details on operation and definitions of the function data structures, typedefs, defines, API data, API structures, and function variables, review the associated *Express Logic User's Manual* listed in the References section.
**This API is only available in NetX Duo DNS Client. For definitions of of NetX Duo specific data types, see the *NetX Duo User Guide* for the Renesas Synergy™ Platform.

Table 2. Status Return Values

Name	Description
NX_SUCCESS	API Call Successful
NX_DNS_NO_SERVER	Client server list is empty
NX_DNS_QUERY_FAILED	No valid DNS response received
NX_DNS_NEED_MORE_RECORD_BUFFER	Input buffer size insufficient to hold the minimum data
NX_PTR_ERROR*	Invalid IP or DNS pointer
NX_CALLER_ERROR*	Invalid caller of this service
NX_DNS_ERROR	Internal error in DNS Client processing
NX_DNS_PARAM_ERROR	Invalid non-pointer input
NX_DNS_CACHE_ERROR	Invalid Cache pointer
NX_DNS_TIMEOUT	Timed out on obtaining DNS mutex
NX_DNS_BAD_ADDRESS_ERROR	Null input address
NX_DNS_INVALID_ADDRESS_TYPE	Index points to invalid address type (e.g. IPv6)
NX_DNS_IPV6_NOT_SUPPORTED	Cannot process record with IPv6 disabled
NX_NOT_ENABLED	Client not configured for this option
NX_DNS_DUPLICATE_ENTRY	DNS Server is already in the Client list
NX_NO_MORE_ENTRIES	No more DNS Servers allowed (list is full)
NX_DNS_SERVER_NOT_FOUND	Server not in client list

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 error codes are only returned if error checking is enabled. For details on error-checking services, see *NetX User Guide* for the Renesas Synergy™ Platform or *NetX Duo User Guide* for the Renesas Synergy™ Platform in NetX *and* NetX Duo, respectively.

3. NetX and NetX Duo DNS Client Module Operational Overview

DNS Messages

The NetX DNS Client module creates the IP instance, enables UDP services in NetX, and initializes the network driver while registering a valid IP address. The module creates an UDP socket for sending and receiving DNS messages to DNS servers listening on port 53 for DNS queries.

To obtain a mapping, the DNS client prepares a DNS query message containing the name or the IP address to be resolved. The message goes to the first DNS server in the server list. If the server has a mapping, it replies to the DNS client using a DNS response message containing the requested mapping. If the server does not respond, the DNS client queries the next server on its list until all its DNS servers have been queried. If no response is received from all its DNS servers, the DNS client retransmits the DNS message starting back at the top of the server list. DNS queries are sent until the retransmission timeout expires. Until a response is received, each iteration down the list of servers doubles the retransmission timeout until the maximum transmission timeout is reached. This timeout is set in the *Maximum duration to retransmit a DNS query (seconds)* property. The default value is 64 seconds; the maximum number of times the DNS client iterates down the server list, which is set by the *Maximum retries for a server* property and defaults to 3.

The typical NetX DNS Client queries are:

- IPv4 address lookups (type A) by using the `nx_dns_host_by_name_get` service.
- Reverse lookups of IP addresses (type PTR queries) to obtain web host names using the `nx_dns_host_by_address_get` service.
- IPv6 address lookups (type AAAA) or IPv4 address lookups (type A), specified in the IP address data type input, in the `nxd_dns_host_by_name_get` service. (This is only available in the NetX Duo DNS Client.)
- Reverse lookups of IP addresses (type PTR queries) to obtain web-host names using the `nxd_dns_host_by_address_get` service. (This is only available in the NetX Duo DNS Client.)

The NetX Duo DNS Client module still supports *nx_dns_host_by_name_get* and *nx_dns_host_by_address_get* services. These are equivalent services, but they are limited to IPv4 network communication, so developers are encouraged to use the *nxd_dns_host_by_name_get* and *nxd_dns_host_by_address_get* services instead.

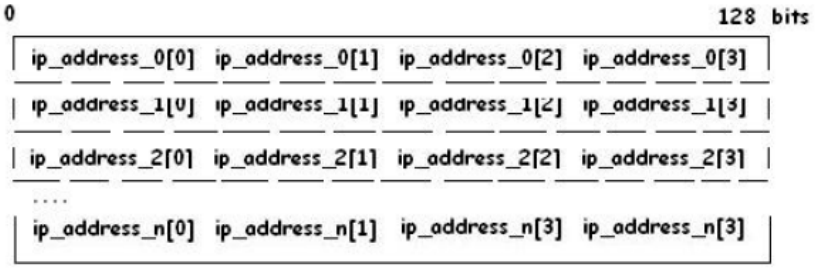
Extended DNS Resource Record Types

If the *Extended RR types support* property is enabled, the NetX DNS Client module also supports the following record type queries:

CNAME	contains the canonical name for an alias
TXT	contains a text string
NS	contains an authoritative name server
SOA	contains the start of a zone of authority
MX	used for mail exchange
SRV	contains information on the service offered by the domain

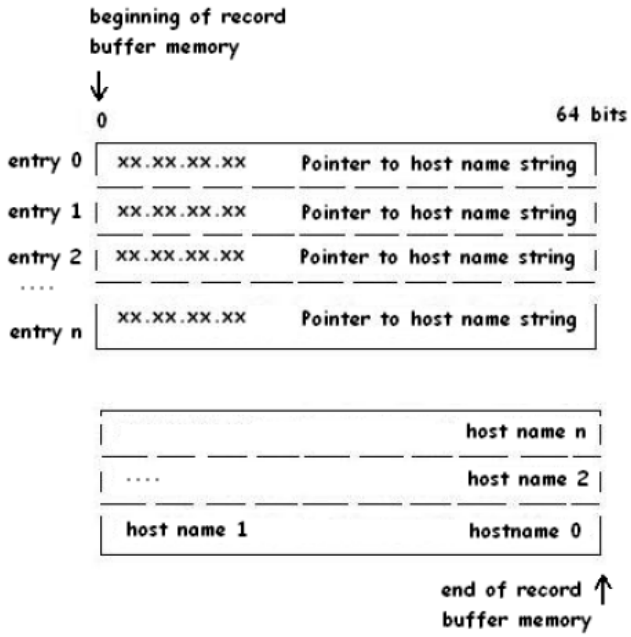
Except for CNAME and TXT record types, the application must supply a 4-byte aligned buffer to receive the DNS data record.

In the NetX DNS Client module, the record data is stored to make efficient use of buffer space. The following example shows a record buffer of fixed length (type AAAA record).



For queries whose record types have a variable data-length (such as NS records with variable-length host names), the NetX DNS Client module saves the data using the following methodology:

- Organizes the buffer supplied in the DNS Client query into fixed-length data and unstructured memory areas.
- Organizes the top of the memory buffer into 4-byte aligned record entries.
- For each record entry, holds the IP address and a pointer to the variable-length data for that IP address.
- Stores variable-length data for each IP address in unstructured area memory starting at the end of the memory buffer.
- Saves variable-length data for each successive record entry in the next area of memory, adjacent to the previous record entries' variable data.
- 'Grows' the variable data towards the structured area of memory holding record entries until there is insufficient memory to store another record entry and variable data (see following figure).



The NetX DNS Client queries use the record storage format to return the number of records saved to the record buffer; this information enables the application to extract NS records from the record buffer. After calling the `nx_dns_domain_service_get` with the receive_buffer pointer to the storage space, the application extracts the SRV data sequentially, taking advantage of the known size of the SRV entry:

```
NX_DNS_SRV_ENTRY *nx_dns_srv_entry_ptr[RECORD_COUNT];

status = nx_dns_domain_service_get(&client_dns,
                    (UCHAR *)"www.my_example.com",
                    &record_buffer[0], BUFFER_SIZE, &record_count,
                    NX_IP_PERIODIC_RATE);

for(i =0; i< record_count; i++)
{
    nx_dns_srv_entry_ptr[i] =
            (NX_DNS_SRV_ENTRY *)(record_buffer + i * sizeof(NX_DNS_SRV_ENTRY));
}
```

DNS Cache

If the *Cache support* property is enabled, the NetX DNS Client module supports the DNS cache feature. The application must set up the DNS cache using the *nx_dns_cache_initialize* service. When caching is enabled, the DNS client checks the DNS cache resource records before sending a DNS query. If it finds the answer in the cache, it directly returns it to the application. Otherwise, it sends out a query message to a DNS server and waits for the reply. When the DNS client gets the response message, it adds a resource record to the cache, if there is a cache entry available.

Each cache entry is a data structure used to hold a resource record. String entries (resource record name and data) in resource records are variable length, and as such, are not stored directly in the resource record. Instead, the resource record sets a pointer to the actual memory location in the cache where the strings are stored. Strings and resource records share the cache. Records are stored from the beginning of the cache and grow towards the end of the cache. String entries start from the end of the cache and grow towards the beginning of the cache. Each string entry has a length field and a counter field. When a string entry is added to the cache, and the same string is already present in the table, the counter value is incremented, and no memory is allocated for the string. The cache is considered full if no more resource records or new string entries can be added to the cache.

3.1 NetX and NetX Duo DNS Client Module Important Operational Notes and Limitations

3.1.1 NetX and NetX Duo DNS Client Module Operational Notes

The NetX DNS Client requires a packet pool for transmitting DNS messages; the packet pool must be before using DNS client services. This can either be the packet pool used by the IP instance (g_packet_pool0), or it can be a separate packet pool added to the project: **X-ware** -> **NetX Duo** -> **NetX Duo Packet Pool Instance** (or **X-ware** -> **NetX** -> **NetX Packet Pool Instance in NetX DNS Client**).

- If the *Use application packet pool* is disabled, the DNS Client application must create the packet pool when the DNS client is created, and then register it with the DNS Client by calling the nx_dns_packet_pool_set service. The module guide project sets this project to enabled, so this is not necessary. The DNS Client shares the same packet pool with the IP instance.
- The *Maximum DNS queries size* property determines the size of the packet payload and defaults to 512 bytes.
- The *Packets in DNS packet pool* property sets the number of packets in the DNS Client packet pool (defaults to 6).

Note: For user-created packet pools, the packet size holds the DNS maximum message size property. The default message size is 512 bytes, plus room for the UDP header (8 bytes), IPv4 header (20 bytes), or IPv6 header (40 bytes), and the network frame header. The Ethernet frame header is rounded up to 16 bytes for 4-byte alignment. This user-created packet pool is only used to send DNS packets. The IP packet pool is set to a 1568-byte packet payload. This packet pool setting works best for the Synergy network driver and the device MTU (typically, 1518 bytes), and it also avoids the need to chain packets in the driver layer.

Before sending DNS queries, the application must add one or more DNS servers to the server list kept by the DNS client. The maximum server list size is set by the *Maximum number of DNS Servers in the Client server list* property.

To add a DNS server, the application can use either the `nx_dns_server_add` service (limited to IPv4 packets) or the `nxd_dns_server_add` service supporting IPv4 and IPv6 packets.

Note In the NetX Duo DNS Client, enabling the *Client has DNS and Gateway Server* property has no effect in the SSP 1.2.0 and earlier. When the DNS client is created, the IP instance does not have a router/gateway address set. Therefore In the NetX DNS Client, the *Client has Gateway Server* can be left disabled.

To explicitly add a DNS server, default gateway server or otherwise, use the *nx_dns_server_add*. To set the default gateway as the server you can type ipconfig in a command line shell and find out what the default gateway address is. This is explained in greater detail in chapter 9.

If the IP instance in your SSP does have the *Default Gateway Address* property, then *Client has DNS and Gateway Server* if enabled will add the default gateway to the DNC Client server list as the primary DNS server.

- To set up the DNS cache, the application can use the `nx_dns_cache_initialize` service with the cache memory buffer pointing to a previously defined buffer. To be notified if the cache is full, use the `nx_dns_cache_notify_set` service. This callback can be ‘disabled’ by clearing the callback using the `nx_dns_cache_notify_clear` service.
- A useful feature is the *Clear previous DNS queries from queue* property; this property enables the DNS client to remove any old DNS server responses from the DNS client receive queue when looking for a response that matches the current query. This means older packets received from previous DNS queries are discarded to prevent the DNS client socket from overflowing and dropping valid packets.

Refer to the following list for NetX DNS services and their NetX Duo DNS equivalents:

NetX DNS API service (**IPv4 only**)	NetX Duo DNS API service (**IPv4 and IPv6 supported**)
nx_dns_host_by_name_get	nxd_dns_host_by_name_get
nx_dns_host_by_address_get	nxd_dns_host_by_address_get
nx_dns_server_get	nxd_dns_server_get
nx_dns_server_add	nxd_dns_server_add
nx_dns_server_remove	nxd_dns_server_remove

3.1.2 NetX and NetX Duo DNS Client Module Limitations

- The DNS client supports one DNS request at a time; threads attempting to make another DNS request are temporarily blocked until the previous DNS request is complete.
- The NetX DNS Client module does not use data from authoritative answers to forward additional DNS queries to other DNS servers.

See the latest SSP Release Notes for any other operational limitations that apply to this module.

4. Including the NetX and NetX Duo DNS Client Module in an Application

To include the NetX and NetX Duo DNS Client module in an application using the SSP configurator, use the following instructions.

Note: It is assumed 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.

To add the NetX DNS Client Module to an application, simply add it to a thread using the stacks selection sequence listed in the following table. The default name for the NetX DNS Client is `g_dns0`; the name can be changed in the associated Properties window.

Table 3. NetX and NetX Duo BSD Support Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_dns0 NetX DNS Client	Threads	New Stack> X-Ware> NetX> Protocols> NetX DNS Client
g_dns0 NetX Duo DNS Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo DNS Client

The following figure shows when the NetX or NetX Duo DNS Client module is added to the thread stack the configurator automatically adds any 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 in common and need only be added once to be used by multiple stacks. Modules with a Pink band may require selecting lower-level drivers, with these drivers either optional or recommended. If lower-level drivers are needed, the user sees "Add" in the module description. Clicking any Pink-banded modules displays the "New" icon and possible choices.

As configured in the figure, the NetX DNS Client module automatically creates its own packet pool. To add a separate packet pool, click the + icon and select **New Stack**> **X-Ware**> **NetX** > **NetX Packet Pool** instance (or **New Stack**> **X-Ware**> **NetX Duo** > **NetX Duo Packet Pool** instance for NetX Duo). In the DNS Client thread entry code, use the nx_dns_packet_pool_set service to set this packet pool (probably g_packet_pool1) as the DNS Client packet pool.

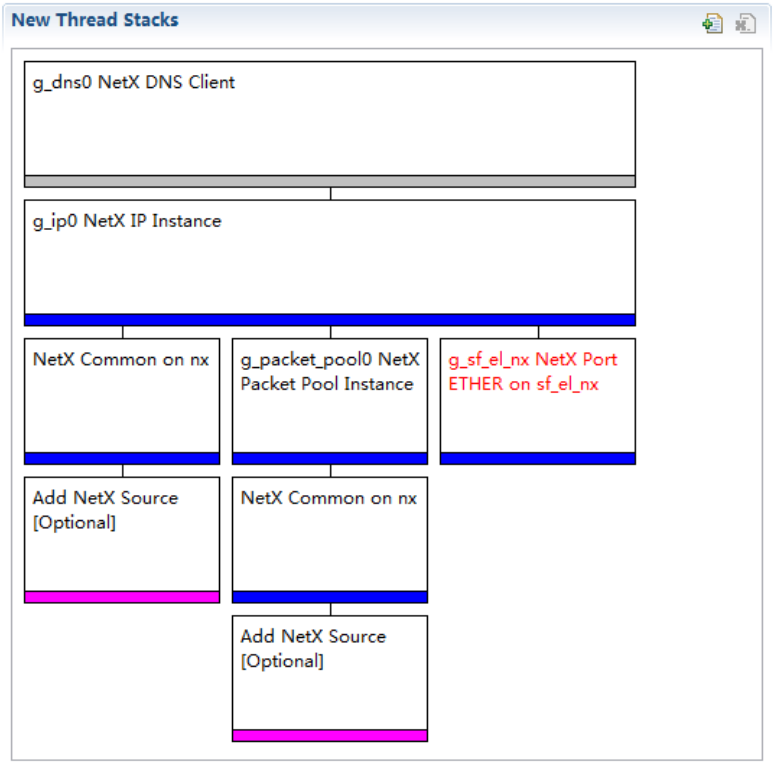


Figure 2. NetX DNS Client Module Stack

5. Configuring the NetX and NetX Duo DNS Client Module

The user configures the NetX and NetX Duo DNS Client Module for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes. These settings must be configured for lower-level modules for successful operation. Only properties that can be changed without conflict are available for modification. Other properties are 'locked' from changes and unavailable. Unavailable properties are identified with a lock icon for the 'locked' property in the ISDE Properties window. This approach simplifies the configuration process and makes it much less error-prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are available from the SSP Configurator Properties tab and are listed in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available in the Properties window of the associated module. Simply select the indicated module and view the Properties window; the interrupt settings are often toward the bottom of the list, so scroll down until they become available. Note that the interrupt priorities listed in the ISDE Properties window indicates the validity of the setting based on the MCU targeted (CM4 or CM0+). This level of detail is not included in the following configuration properties tables but is easily visible within the ISDE when configuring interrupt-priority levels.

Note: You may want to open your ISDE and create the NetX DNS Client module and explore the property settings in parallel with looking over the following configuration table settings. This help to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Table 4. Configuration Settings for the NetX and NetX Duo DNS Client Module

ISDE Property	Value	Description
DNS Control Type of Service	Normal, Minimum delay, Maximum data, Maximum reliability, Minimum cost Default: Normal	DNS control type of service selection
Time to live	128	Time to live selection
Client DNS IP version (**Not included in NetX)	IPv4, IPv6 Default: IPv4	Client DNS IP version selection
Maximum number of DNS Servers in the Client server list	5	Maximum number of DNS Servers in Client server list selection
Maximum DNS queries size (bytes)	512	Maximum DNS queries size selection
Packets in DNS packet pool (units)	6	Packets in DNS packet pool selection
Maximum retries for a server	3	Maximum retries for a server selection
Maximum duration to retransmit a DNS query (seconds)	64	Maximum duration to retransmit a DNS query selection
Packet allocate timeout (seconds)	2	Packet allocate timeout selection
Client has DNS and Gateway server	Enable, Disable Default: Disable	If enabled, the default gateway address is set as the DNS server. In SSP 1.2.0 there is no effect See explanation in Section 3.1.1
Use application packet pool	Enable, Disable Default: Enable (for NetX Duo) Default: Disable (for NetX)	Use application packet pool selection
Clear previous DNS queries from queue	Enable, Disable Default: Disable	Clear previous DNS queries from queue selection
Extended RR types support	Enable, Disable Default: Disable	Extended RR types support selection
Cache support	Enable, Disable Default: Disable	Cache support selection
Name	g_dns0	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.

In some cases, settings other than the defaults for lower-level modules can be desirable. For example, it might be useful to select different MAC or IP Addresses. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are intuitive to configure and usually can be determined by inspection of the associated properties window from the SSP configurator.

5.1 Configuration Settings for the NetX DNS Client Lower-Level Modules

Typically, only a small number of settings must be modified from the default for lower-level modules, as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module. You must ensure that entries highlighted in yellow will work with your environment.

Table 5. Configuration Settings for the NetX IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection. This address must be on the same local network as the default gateway. The value listed is the default value
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
**IPv6 Global Address (use commas for separation)	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection; not necessary if IPv6 is not used. Only applicable in NetX Duo DNS Client, not NetX DNS Client
Default Gateway Address (use commas) *This property is not available in SSP 1.2.0.*	192,168,0,1	IP address of the local host network default gateway.
**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. This is set automatically based on the network interface MAC address unless you specify a different address here.
IP Helper Thread Stack Size (bytes)	Recommend: 2048 Default: 1024	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	Recommend: 1 Default: 3	IP Helper Thread Priority selection. The IP thread should be the highest priority of all threads using NetX services for optimal performance
ARP	Enable	ARP selection
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	Enable, Disable Default: Enable	TCP selection
UDP	Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection

Notes:

- **Indicates properties only included with NetX Duo.
- 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 6. Configuration Settings for the NetX Common Instance

ISDE Property	Value	Description
No configurable properties		

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 7. Configuration Settings for the NetX Packet Pool Instance

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	Default: 640	Packet size selection. If this is shared by the DNS Client, this is plenty large enough for most DNS packets. If the DNS packet fails to receive server DNS packets, try increasing the size. A Wireshark packet trace is a helpful tool for knowing what size packets the DNS Client is getting.
Number of Packets in Pool	Default: 16	Number of packets in pool selection. The default value is adequate for the IP instance. If this packet pool is shared with the DNS Client, and there is heavy network traffic, increasing the number of packets may help improve performance.

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 8. Configuration Settings for the NetX Port ETHER

The entries in yellow must be changed to work properly with the SK-S7G2 board.

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled 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	Default IOPORT_PORT_07_PIN_06 Set to: IOPORT_PORT_08_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), Disabled Default: Disabled	Ethernet interrupt priority selection. Any priority level will suffice for this project, but do not leave it disabled.
Name	g_sf_el_nx	Module name
Channel	Default: 0 Set to: 1	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 9. Configuration Settings for the NetX Common Instance

ISDE Property	Value	Description
No configurable properties		

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 and NetX Duo DNS 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 and NetX Duo DNS Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU to communicate to external devices. I/O pins are selected and configured as required by the external device. The following table lists the pin selection method within the SSP configuration window.

Note: The operating mode selected determines the peripheral signals available and the MCU pins required.

Table 10. 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 11. Pin Configuration Settings for the ETHERC1

Property	Value	Description
Pin Group Selection	Mixed, _A only (Default: _A only)	Pin group selection
Operation Mode	Disabled, Custom, RMII (Default: Disabled)	Select RMII as the Operation Mode for ETHERC1
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 MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

6. Using the NetX and NetX Duo DNS Client Module in an Application

The following example assumes the NetX system is already initialized. Use these typical steps when implementing the NetX DNS Client module in an application:

1. Wait for valid IP address using the `nx_ip_status_check` API.
2. Enable DNS caching by calling the `nx_dns_cache_initialize` [optional].
3. Add one or more servers to the client list using the `nx_dns_server_add` API.
4. Send DNS name query using `nxd_dns_host_by_name_get` API to obtain an IP address.
5. Extract DNS server resource records using known size and arrangement of records packaged by the DNS client.

The following diagram shows these common steps a in a typical operational flow.

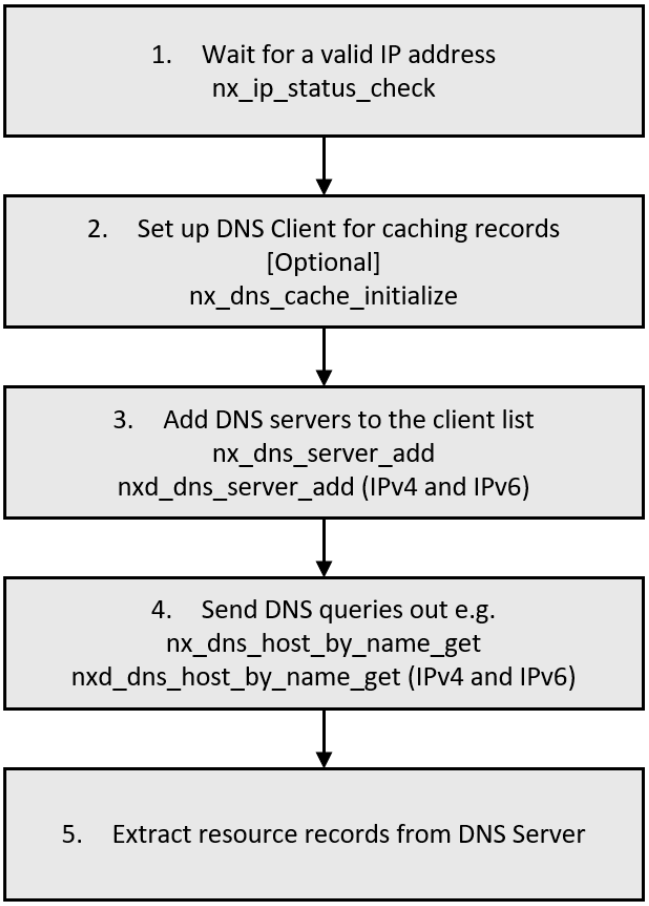


Figure 3. Flow Diagram of a Typical NetX DNS Client Module Application

7. The NetX and NetX Duo DNS Client Module Application Project

The application project associated with this module guide demonstrates the steps needed in a full design. The project can be found using the link provided in the References section. You may want to import and open the application project within the ISDE and view the configuration settings for the NetX DNS Client module. You can also read over the code (in `dns_client_thread_entry.c`) that shows the NetX DNS Client module APIs in a complete design.

The application project demonstrates the typical use of the NetX DNS Client module APIs. It sets the DNS Client packet pool and adds a DNS Server IP address to its server list; it then sends a query to get the IP address for a website. If it succeeds, it prints the list of IP addresses parsed from the server response(s).

Table 12. 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

The following figure shows a simple flow of the application project client and server functions.

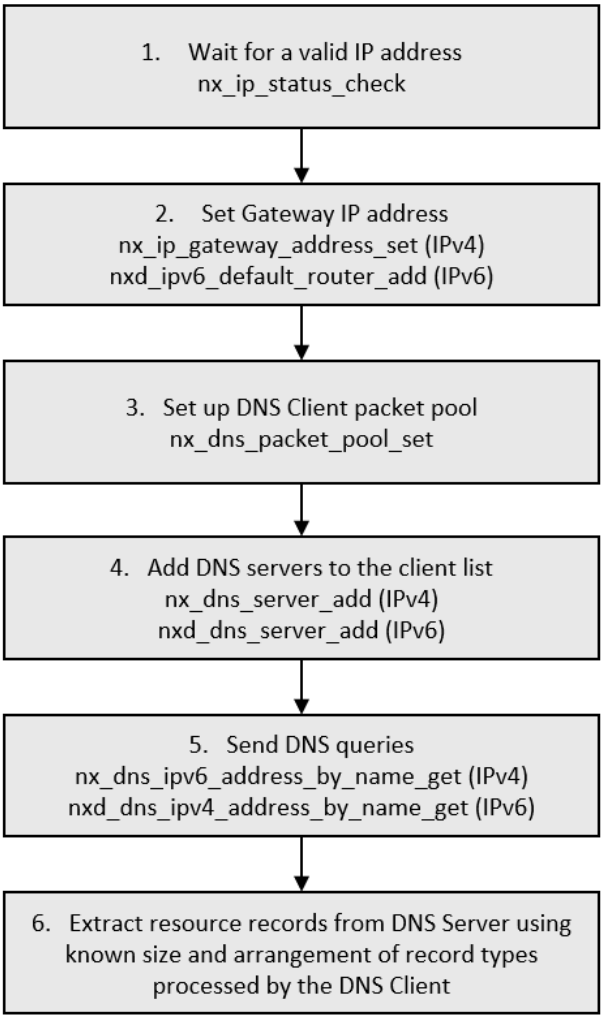


Figure 4. NetX and NetX Duo DNS Client Module Application Project

The `dns_client_thread_entry.c` file is located in the project once it has been imported in the ISDE. The `dns_client_thread_entry` function calls the `run_dns_client_session` function in the `DNS_Client_MG_AP.c` file which actually performs the DNS query. You can open these files within the ISDE, read the description, and follow along to help identify key uses of the APIs provided.

The first section of `dns_client_thread_entry.c` lists the header files, `dns_client_thread.h` and `DNS_Client_MG_AP.h`. It also includes `studio.h` and a code section which allows semi-hosting to display results using `printf()`. `DNS_Client_MG_AP.h` in turn lists the `nxd_dns.h` (or `nx_dns.h`) for NetX DNS Client which references the DNS client-instance structure.

The `dns_client_thread_entry.c` sets up the DNS server address, the buffer for receiving the DNS server data, and the size of the buffer; it also defines the hostname to query on.

The next section in `dns_client_thread_entry.c` is the entry function for the main program-control section. The code checks that the IP instance and driver are properly initialized, and the IP instance has a valid IP address. Then it sets the gateway address for communicating with external hosts outside the current network segment using the `nx_ip_gateway_address_set` API. This is not required to run this project if the DNS server resides on the local network. Note this only applies for IPv4 networks. For IPv6 networks, it sets the default router using the `nxd_ipv6_default_router_add` API. Next, the thread entry function sets the DNS client packet pool, which in this case is shared with the IP instance default packet pool, `g_packet_pool0`. This packet pool is specifically used by the DNS client to send queries. Finally, it sets up the DNS server addresses, including a DNS IPv6 server address if USE_IPV6 is defined in `DNS_Client_MG_AP.h`. In NetX Duo, the **NXD_ADDRESS** type is preferred since it supports IPv4 and IPv6 addresses. In NetX, IP addresses are stored in the **ULONG** data type.

At this point the application is ready to run a DNS client session and send queries out. It calls the `run_dns_client_session` function, which adds the input server address to the DNS client list of servers,

then sends a DNS query out with the input host name. It returns a buffer pointing to the list of records extracted from the DNS server response (and the number of records) if no errors occur. If an error occurs, it returns the number of errors as the 'status' return.

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

A few key properties are configured in this application project to support the required operations and the physical properties of the target board and MCU. The properties with the values set for this specific project are listed in the following tables. You can also open the application project and view these settings in the Properties window as a hands-on exercise.

Table 13. NetX DNS and NetX Duo DNS Client Module Configuration Settings for Application Project

ISDE Property	Value Set
Use application packet pool	Enable
Name	g_dns0

Table 14. NetX and NetX Duo Packet Pool Configuration Settings for Application Project

ISDE Property	Value Set
Packet Size (bytes)	1568
Name	g_packet_pool0

8. Customizing the NetX and NetX Duo DNS Client Module for a Target Application

Some configuration settings are normally changed by the developer from those shown in the application project. For example, the application can make alternative queries such as an MX query to obtain a mail exchanger server that accepts delivery of email for the specified domain.

```
/* Send DNS MX query, and record the multiple mail exchange info. */
status = nx_dns_domain_mail_exchange_get(&client_dns, (UCHAR *)"google.com",
                  &record_buffer[0], 500, &record_count, 4 * NX_IP_PERIODIC_RATE);

/* Check the record buffer.  */
nx_dns_mx_entry_ptr1 = (NX_DNS_MX_ENTRY *)record_buffer;
nx_dns_mx_entry_ptr2 = (NX_DNS_MX_ENTRY *)(record_buffer +
                       sizeof(NX_DNS_MX_ENTRY));
nx_dns_mx_entry_ptr3 = (NX_DNS_MX_ENTRY *)(record_buffer + (2 *
                       sizeof(NX_DNS_MX_ENTRY)));
nx_dns_mx_entry_ptr4 = (NX_DNS_MX_ENTRY *)(record_buffer + (3 *
                       sizeof(NX_DNS_MX_ENTRY)));
nx_dns_mx_entry_ptr5 = (NX_DNS_MX_ENTRY *)(record_buffer + (4 *
                       sizeof(NX_DNS_MX_ENTRY)));
```

You can customize the DNS client application further by enabling the *Clear previous DNS queries from queue* property. If this property is enabled, the DNS client removes older DNS server responses from the DNS client receive queue until it finds a response that matches the current query. This prevents the DNS client socket from overflowing and dropping newer (valid) packets.

Another common customization is to create a separate packet pool specifically for the DNS client to send packets. This requires setting the *Use Application packet pool* property to disabled; and a packet pool to be added to the DNS thread stacks pane. Click the (**+**) icon -> **X-ware** -> **NetX** -> **NetX Packet Pool Instance** or, if using NetX Duo DNS Client, **X-ware** -> **NetX Duo** -> **NetX Duo Packet Pool Instance**.

Lastly, the NetX and NetX Duo DNS Client modules can enable caching by calling `nx_dns_cache_initialize` with a pointer to cache memory and cache size as inputs. This requires setting the *Cache support* property of the DNS Client instance. The DNS client stores the DNS server response data. On subsequent DNS client queries, the DNS client first checks the cache if the information already exists.

9. Running the NetX and NetX Duo DNS Client Module Application Project

To run the NetX DNS Client module application project and see if it executed on a target kit, you can simply import it into your ISDE, compile, and run debug. See the *Renesas Synergy™ Project Import Guide* (included in the package) for instructions on importing the project into e^2 studio or IAR EW for Synergy to build and run the application.

To implement the NetX DNS Client module application in a new project, follow the steps to define, configure, auto-generate files, add code, compile, and debug on the target kit. Following these steps is a hands-on approach to help make the development process with SSP practical, while just reading over this guide tends to be more theoretical.

Note: The following steps are detailed sufficiently for someone experienced with the basic flow through the Renesas Synergy development process. If these steps are unfamiliar, see the *SSP User's Manual* for guidance.

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

1. Create a new Renesas Synergy project for the S7G2-SK called DNS_CLIENT_AP.
2. Select the **Threads** tab.
3. Click the **+** icon in the Threads panel.
4. Set the stack size to **2048** and set the thread instance to **dns_client_thread**.
5. In the thread stack pane, click the **+** icon and select **X-Ware** -> **NetX Duo** -> **Protocols** -> **NetX Duo DNS Client** (or **X-Ware** -> **NetX** -> **Protocols** -> **NetX DNS Client for NetX application**).
6. Set the properties for the DNS Client instance, IP instance, and the NetX Port ETHER driver.
   Be sure to set the **Ethernet Interrupt Priority**; its default value is **Disabled**. For the SK-S7G2 board, select **Channel 1**. The Channel 1 Phy Reset Pin should be **IOPORT_PORT_08_PIN_06**. These properties default to **0** and **IOPORT_PORT_07_PIN_0**6.
7. Set the IPv4 address (and IPv6 if using IPv6 network) of the IP instance, as follows:
   — Get the IP address assigned to your board: type `ipconfig` at the command line of a PC on the local network. Use the PC's network address to create an IP address for the board. For example, if the ipconfig command lists 192.2.2.201 as the PC's IP address, create IP address 192.2.2.x, where x is any value between 2 and 254.
   — Now check whether that address is in use by sending a ping command. If there is no response, you can use this IP address. You can also type **arp -r** for assigned IP addresses on the network. Pick an address NOT listed!

```
C:\Perl64\eg>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet 4:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2001:db8::201
   Link-local IPv6 Address . . . . . : fe80::2c1b:8bd0:6434:da1b%19
   IPv4 Address. . . . . . . . . . . : 192.2.2.201
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.2.2.1

C:\Perl64\eg>ping 192.2.254

Pinging 192.2.0.254 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.2.0.254:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

8. If you would like to make the network router the DNS server, and the IP component has a *Default Gateway Address* property (not available in SSP 1.2.0), be sure to set that property to the router address and set the DNS Client *Client has DNS and Gateway Server* property to enabled. The router address/gateway is also listed in the ipconfig output above (192.2.2.1).
9. Alternatively, you can define SERVER_IP_ADDRESS at the top of dns_client_thread_entry.c to set the DNS server IP address. If you have set *Client has DNS and Gateway Server* to enabled (and the IP component has a *Default Gateway Address* property) this adds that DNS server to the DNS Client list of servers.

Notes:

1. The DNS Client can have more than one DNS server, but in the interests of simplicity, this demonstration project only registers one DNS server IP address with the Client.
2. The local router/gateway and SERVER_IP_ADDRESS — whichever (or both) that you use — should be on the same network as your DNS Client.
3. When using the DNS Client with IPv6, define **USE_IPV6 in DNS_Client_MG_AP.h** and set the server IPv6 address to your DNS server IPv6 address. Note that the Client has DNS and the Gateway Server has no effect on IPv6 DNS clients. For example:

```
server_duo_address.nxd_ip_address.v6[0] = 0x20010db8;
server_duo_address.nxd_ip_address.v6[1] = 0x0;
server_duo_address.nxd_ip_address.v6[2] = 0x0;
server_duo_address.nxd_ip_address.v6[3] = 0x1;
```

4. Click the **Generate Project Content** button.
5. Set optimization to **none -O2** (should default to this value):
   - Right-click the project and choose Properties -> **C/C++ Build** -> **Settings** -> **Optimization**. Set the Optimization Level to **None (-O2)**.
   - For optimal debugging, at the expense of performance, you can set this to **-O0**.
6. Enable debug output [optional].
   A. Right click the project and choose **Properties** -> **C/C++ Build** -> **Settings** -> **Cross ARM C Compiler** -> **Preprocessor**.
   B. Add a define in the Defined symbols pane by clicking on the **+** icon. Type **SEMI_HOSTING**.
7. For NetX Duo DNS Client applications, define NETX_DUO in the project preprocessor list:
   A. Right click the project and choose **Properties** -> **C/C++ Build** -> **Settings** -> **Cross ARM C Compiler** -> **Preprocessor**.
   B. Add define in the Defined symbols pane by clicking the **+** icon. Type **NETX_DUO**.
8. Right-click the project and choose **Build Project**.
9. Right-click the project and choose Debug as -> **Renesas GDB Hardware Debugging**
10. Connect to the host PC to the board via Ethernet (not Wifi) via a micro-USB cable to J19 on SK-S7G2. The PC should not be connected to another network interface such as Wifi.
11. Run the application.
    If the return value from the run_dns_client_session service is zero, print out the results of the query (the IP address may vary):

```
Response to www.berkeley.com query: record_count = 1
record 0: IPv4 address: 184.168.221.104
```

Figure 5. Example Output from NetX and NetX Duo DNS Client (IPv4) Application Project

If **USE_IPV6** is defined on the application (NetX Duo DNS Client only), the results should look like this:

```
Response to www.berkeley.com query: record_count = 1
record 0: IPv6 address: 0x2607f140 0x00000081 0x00000000 0x0000000f
```

Figure 6. Example Output from NetX and NetX Duo DNS Client Application Project

10. NetX and NetX Duo DNS Client Module Conclusion

This module guide has provided all 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. The Renesas Synergy Platform makes these steps much less time consuming and removes the common errors, like conflicting configuration settings or incorrect selection of lower-level drivers. The use of high-level APIs (as demonstrated in the application project) illustrate additional 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.

11. NetX and NetX Duo DNS Client Module Next Steps

After you have mastered a simple DNS Client project, you may want to review a more complex example.

The DNS Client has many other types of queries, especially if RR record support is enabled. This lets the application make SOA, MX, PTR, CNAME and other queries using similar API. Note that the DNS Client module takes care of packet handling and releasing packets back to the packet pool.

You may also find that setting a DHCP Client is a better fit for your target application than setting a static IP. The *NetX™ DHCP Client Module Guide* shows how to use it to obtain a dynamic provided IP address from a DHCP Server.

12. NetX and NetX Duo DNS Client Module Reference Information

The *SSP User Manual* is available in HTML in the SSP distribution package, and as a pdf from the Synergy Gallery (www.renesas.com/synergy/software).

Links to all the most up-to-date NetX and NetX Duo Module Guide Resources are available on the Synergy Knowledge Base: https://en-support.renesas.com/search/NetX%20DNS%20Client%20Module%20Guide%20Resources.

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan.12.18	—	Initial Release
1.01	Jan.04.19	—	Minor changes and correction throughout

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.