Renesas Synergy™ Platform

NetX™ and NetX Duo™ Source Module Guide

Introduction

The NetX™ and NetX Duo™ source modules allow the developer to modify some of the key properties that control NetX operations. This document provides an easy reference for the NetX and NetX Duo source module in the Renesas Synergy™ e^2 studio ISDE. The properties are explained in greater detail than the footer comment supplied with each property. **If** and **when** to change a default value is included in the context specific usage. This document should make it easier to use the NetX and NetX Duo source component without having to cross reference with the *Express Logic NetX and NetX Duo User's Guide*, as well as help you quickly get familiar with NetX and NetX Duo features. Note that there is no Application Project associated with this Module Guide.

Adding the NetX or NetX Duo source component enables you in the Synergy configurator environment to customize the NetX and NetX Duo libraries, change values from default settings, and enable or disable certain features. Otherwise they must use the prebuilt NetX or NetX Duo library. In most projects beyond simple socket programs, you will typically want to customize their NetX or NetX Duo environment. Note that the ThreadX® source component is automatically added when adding a NetX or NetX Duo source component.

Without adding the NetX or NetX Duo source component, the Synergy ISDE configurator will use a prebuilt library with NetX and NetX Duo default settings.

Contents

1. NetX and NetX Duo Module Source Module Features

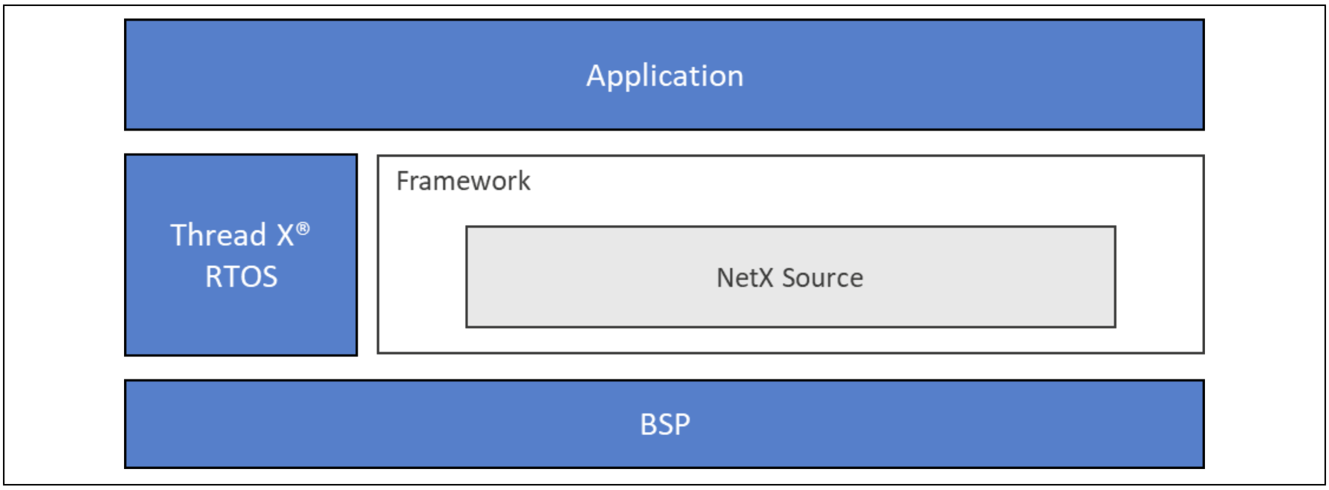

Figure 1. NetX and NetX Duo Source Module Block Diagram

2. NetX and NetX Duo Source Module APIs Overview

There are no APIs associated with the NetX or NetX Duo source module. This module is used to configure various NetX or NetX Duo properties. Note that lower level drivers may return Common Error Codes. See the associated module in the *SSP User's Manual* API References for a definition of all relevant status return values.

3. NetX and NetX Duo Source Module Operational Overviews

Using the NetX or NetX Duo source module is a bit different than using other SSP modules. The NetX or NetX Duo source module is used to configure networking operations; it doesn't provide API functions, callbacks or other typical module functions. There is no typical operational overview of the NetX or NetX Duo module. Refer to the *NetX User's Manual*, available from the Synergy Gallery for the operational details of NetX. The main functions this module guide is concerned with are the configurable properties as described in Chapter 5.

3.1 Using TraceX® with NetX and NetX Duo

If TraceX® is enabled in the ThreadX source component, is automatically included when adding NetX or NetX Duo source components, the project containing the ThreadX and NetX, or NetX Duo library must be rebuilt or TraceX macros that log events will not be executed.

4. Including the NetX and NetX Duo Source Module in an Application

A network project generated in the e^2 studio configurator will automatically include an object **Add NetX Source**.

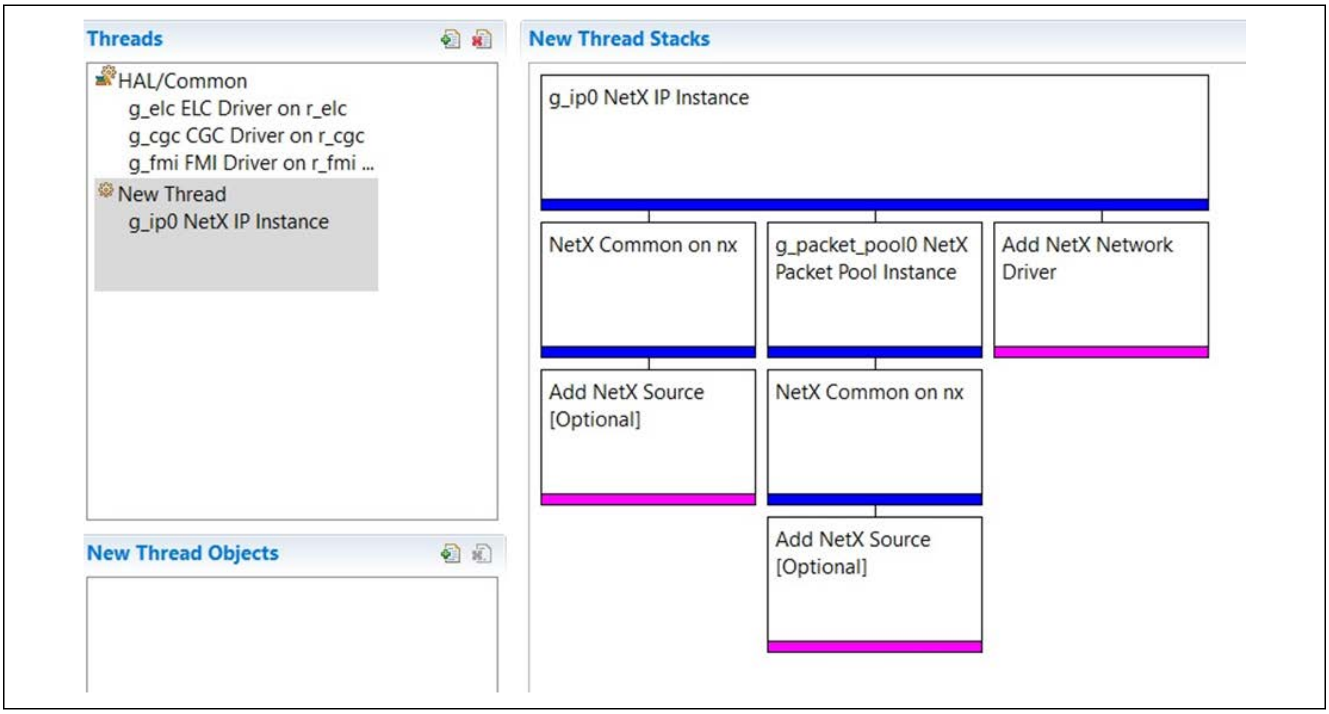
Threads
HAL/Common
g_elc ELC Driver on r_elc
g_cgc CGC Driver on r_cgc
g_fmi FMI Driver on r_fmi ...
New Thread
g_ip0 NetX IP Instance
New Thread Objects
New Thread Stacks
g_ip0 NetX IP Instance
NetX Common on nx
g_packet_pool0 NetX Packet Pool Instance
Add NetX Network Driver
Add NetX Source [Optional]
NetX Common on nx
Add NetX Source [Optional]

Figure 2. NetX and NetX Duo Source Module Stack

To add the NetX or NetX Duo source component to a project, click on the **Add NetX Source** (optional), or **Add NetX Duo Source** (optional object) for NetX Duo, in the e^2 studio ISDE configurator, then choose **New**. If Multiple Add NetX Source boxes are there, all are updated automatically. Note: a ThreadX source component is added automatically.

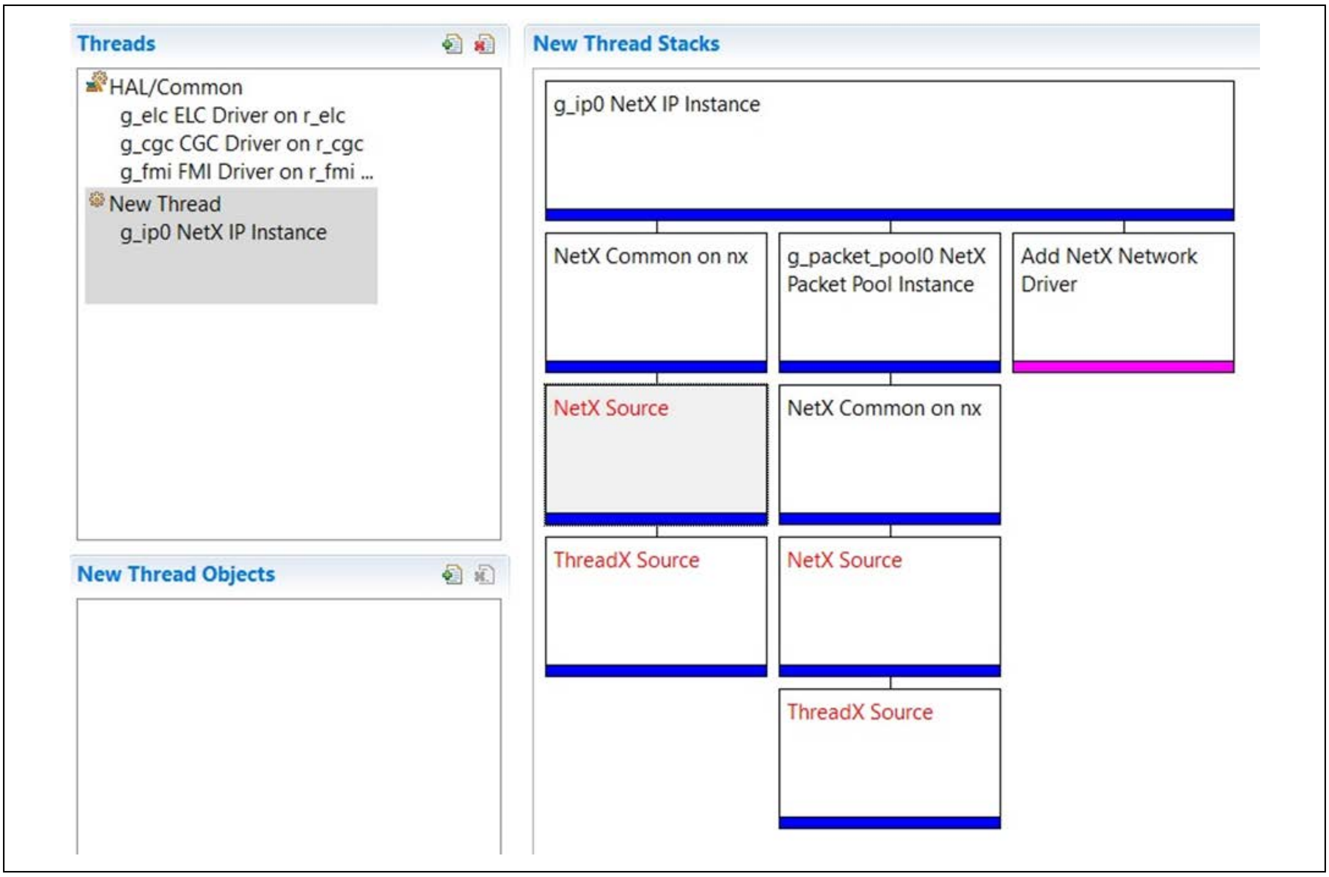

Figure 3. ThreadX Source Component is Added Automatically

5. Configuring the NetX and NetX Duo Source Module

This section describes the configurable properties available to the NetX or NetX Duo source module and describes the cases in which they can be changed from the default values to customize NetX or NetX Duo operations.

Note: After changing NetX and NetX Duo property settings, the developer must click the **Generate Project Content** button to update the project configurator in the ISDE. The NetX or NetX Duo library **must** then rebuild the project. Changing a property (or applying a #define in the preprocessor list), without rebuilding the project, will not affect any changes. The Synergy™ Software Package (SSP) ISDE will use the previously built library.

5.1 NetX or NetX Duo Source Module Configurable Properties

The following list of properties may not match the order in the NetX or NetX Duo property table. The properties are grouped in general categories; ARP, TCP, IGMP, and so on, for easy reference.

5.1.1 A Few Notes about the NetX or NetX Duo Properties

These are some important notes developer should understand before exploring details for many of the properties available in the NetX or Net Duo source module.

Default Settings	Most default settings are derived from RFC recommendations for the protocol relevant to the property.
IP Helper Thread Stack Size (bytes) – default value 1024	Stack size of the IP thread task that processes application NetX API calls, periodic events, and deferred ISR events; packets received. There are some cases where a larger stack size will be required. Usually, the optimal stack size is determined empirically for an individual application.
IP fragmentation – default value enabled	Use the NetX/NetX Duo Source Fragmentation Option property to enable/disable fragmentation. This property is being deprecated in the IP instance component.
IP Helper Thread Priority – default value 3	Priority at which the IP thread task operates. Under some circumstances, it is advisable to increase to 1 (highest valid priority). For example, if there are multiple network application thread tasks (DNS, HTTP, DHCP) in a project, then an IP helper thread at a higher priority (higher than network application threads) can service all network operations with higher priority, effectively improving responsiveness of the application thread tasks.

5.1.2 IP Layer Properties

Error Checking – default value enabled	Generally, error checking is enabled during development and debugging, and disabled when building a release version. When enabled, NetX and NetX Duo include error-checking services that check input and other parameters before calling the actual API. Some things checked include: • NULL pointer input • Invalid non-pointer parameters, such as an invalid IP address type or an IP address equal to zero • Enables for required configurable options; such as TCP, which is enabled to use the `nx_tcp_socket_receive` API. • NetX or NetX Duo data structure IDs which must match what is expected; — `ip_ptr -> nx_ip_id == NX_IP_ID /*` check IP instance structure */ • Size of the data structure; the IP instance matches the size of the data structure in the NetX library. These last two checks guard against an application using a different version of the NetX or NetX Duo library than the application is using.

Static Routing – default value disabled	Allows certain destinations to be routed through a specific router. Normally, a packet being transmitted is routed through the IP instance gateway/router as the next hop. When static routing is enabled, NetX checks the static routing table to determine if a packet's **next hop** address should go through a specific router, rather than the IP instance gateway. The static routing table is limited to `NX_IP_ROUTING_TABLE_SIZE` entries, and there are various API such as `nx_ip_static_route_add` for managing the routing table. This is not commonly used but there are certain situations where it is necessary.
Physical Header and Physical Trailer – default value 16 and 4 bytes, respectively	The type of network medium determines the physical (sometimes called a frame) header. An Ethernet network has a header of size 16 bytes. Wi-Fi has a larger header. The physical trailer is currently unused for Ethernet but may need to be configured, depending on the physical network used by the application. If the physical header is not set correctly, packets will not be assembled and most NetX packet send and receive services will not function correctly.
Maximum Listen Requests – default value 10	Value that is used by TCP server applications to define the TCP listen queue size. When a TCP server socket binds a port, via API `nx_tcp_server_socket_listen`,, it calls a listen request on the IP instance. When it unlistens on that port, via API `nx_tcp_server_socket_unlisten`, the listen request is released, and the port is available to listen by another socket. The default value is not usually changed.
Driver Deferred Processing – default value enabled	Enables the IP thread task to defer handling of the packet receive interrupts to the context of the IP thread task. Otherwise, a packet receive event is handled in the context of the interrupt. This might result in a faster response to a specific packet but makes the overall system performance slower.
Loopback Interface – default value enabled	If enabled, NetX creates a loopback interface to send and receive packets to itself. Note: this is not counted as a physical interface. An application needs to consider the number of network interfaces if they are using multiple physical interfaces. The loopback interface is not counted as a physical network interface (see *Maximum Physical Interfaces*).
Maximum Physical Interfaces – default value 1	One network interface is the default. It is typically called the primary interface. The IP instance keeps a table of interfaces. The primary interface is at index 0. For secondary interface(s), the Maximum Physical Interface value should be incremented by one for each secondary interface. For multiple interface use, the typical scenario is a router; with a local or private network on one interface, and a global or public interface on the other. Another scenario is a device with two different network interfaces; such as Ethernet and Wi-Fi. To attach secondary interfaces to the IP instance, use the `nx_ip_interface_attach` API. When an interface is not referenced in an API, the action is on the primary interface. For example, `nx_ip_status_check` operates on the primary interface, while `nx_ip_interface_status_check` performs the operation on a specific interface. **Note**: Some developers use multiple IP instances to handle multiple physical networks. Strongly not recommended to create multiple IP instances!
NAT (available in NetX Duo only) – default value disabled	NAT is a protocol that allows NetX Duo to map packets between the local/private and public/global networks, acting as a kind of router. To do this, the NetX Duo must be enabled to forward packets from a local host onto the global network with the host's global IP address and a port that the NetX Duo host determines. When NAT is enabled on the IP instance, NetX Duo can now forward packets in the manner. Note: NAT requires **Maximum Physical Interfaces** to be set to 2. Note: NAT is available only in NetX Duo.

Fragmentation option – default value enabled

Fragmentation occurs when the packet data exceeds the device MTU (max. transfer unit), typically 1518 bytes, including the frame header for Ethernet networks. Whenever possible, an application should avoid fragmentation at the IP layer, although there is not much one can do on the receiving side.

Another aspect of fragmentation to consider is the potential to deplete the packet pool used to receive packet fragments. When NetX is receiving many packet fragments, the packet pool used to receive these packets can be easily depleted, since packets cannot be returned to the packet pool until the entire packet is assembled by the IP layer and forwarded to the application. This situation can be remedied by increasing the number of packets in the packet pool at startup time (cannot be done at run time), if the board has the memory required.

Note: This should not be confused with segmentation performed at the TCP layer. See TCP MSS Minimum for more details on segmentation.

Packet Header Pad Size – default value 0 (no padding)

Only used in NetX to increase the size of the NX_PACKET structure for alignment purposes. It has no effect on the default value. For Packet Header pad size >= 1, the NX_PACKET STRUCT is increased by the size of the nx_packet_pad array:

```
typedef  struct NX_PACKET_STRUCT
{
     ….
  #ifdef NX_PACKET_HEADER_PAD
    /* Define a pad word for 16-byte alignment, if
necessary.  */
    ULONG nx_packet_pad[NX_PACKET_HEADER_PAD_SIZE];
  #endif
}
```

By default, it is not defined. It is outside the scope of this document to make recommendations for defining padding packet alignment for Synergy projects.

Checksums – default value enabled

In NetX and NetX Duo, checksums for incoming (RX) and outgoing (TX) packets can be separately enabled or disabled. Checksums should not be disabled, unless the RFC for that network protocol (UDP, TCP, or ICMP) indicate a zero checksum is allowable. Checksums generally need to be enabled. Disabling checksums might increase throughput but may also cause packets to be dropped by the other side of a connection.

Extended Notify Support – default value disabled

When enabled, NetX will notify the application of various events related to TCP socket connections, above and beyond what NetX normally does (such as the callbacks specified in the `nx_tcp_socket_receive_notify` and `nx_udp_socket_receive_notify` APIs).

Extended Notify Support is required by NetX and NetX Duo BSD sockets.

The APIs enabled for this feature are listed below. They allow NetX to notify the application when NetX receives a TCP connection request, a TCP connection is completed, a TCP disconnect on a socket is completed, and to set the TCP socket state in the timed wait state. This is necessary for a TCP to operate in a non-blocking context.

```
nx_tcp_socket_syn_received_notify
nx_tcp_establish_notify
nx_tcp_disconnect_complete_notify
nx_tcp_timed_wait_callback
```

Source Address Check – default value disabled

This checks all incoming packets for an invalid IP address. Specifically, it checks if 1) the IP address bits masked with the network mask does not equal the 1s complement of the network mask; 0xFF, 2) the IP address is not zero and the unmasked address bits do not equal zero, and 3) the address is not type D address (0xE0000000). This extra bit of processing at the IP level in NetX incurs a small performance penalty.

5.1.3 IGMP/Multicast Properties

Maximum multicast groups – default value 7	Defines the size of the multicast table in effect to set the limit on the number of multicast groups that can be joined. Generally, not modified.
IGMPV – default value enabled	IGMPv2, unlike IGMPv1, allows group-specific join queries, leaves messages, and has a method to determine which router will forward queries (instead of all IGMP routers on the network).

5.1.4 ARP Properties

ARP Cache size in Bytes – default value 520	Defines the size of the ARP cache. This table holds all the ARP entries. If a table is full, no more ARP entries can be added till existing entries 'age' (see explanation of ARP Expiration Rate in this section) and are removed. There are some configurable options listed with the NetX/Duo Source element that can affect the number of ARP entries added to the table, such as ARP Auto ARP Entry and ARP Expiration Rate defined in this section. The ARP cache size may need to be increased if the node is expected to communicate with large amount of the nodes on its local network.
ARP Auto ARP Entry – default value enabled	This allows NetX to add an ARP entry when an ARP packet is received for which the table has no matching IP address. This will happen regardless if the response was directed to/from the NetX device. The idea behind auto ARP entry is to increase the efficiency of NetX by reducing the need to send ARP queries if the data is already in its ARP table. The downside is the ARP table can fill up and no more new ARP queries can be added. To prevent this, disable this feature. When disabled, NetX will only save ARP entries if the request was generated by NetX or directed to the NetX device. This option is not required for most applications. Alternatively, one can leave the Auto ARP Entry enabled, and set the expiration rate to non-zero to guarantee the entries will expire (and therefore be deleted) if not used. See **ARP Expiration Rate** for more details on the table entry expiration in this section.
ARP Expiration Rate – default value 0 seconds	By default, ARP entries in NetX are **static**. They have no expiration rate and thus do not age. Otherwise, as an ARP entry ages, its timeout value decreases. When the timeout value reaches zero, the ARP entry is removed. If a new query pertaining to that ARP entry is received, the timeout value is reset. The timeout is also reset if the entry is accessed by the IP layer to transmit a packet. To determine if entries are being aged/deleted, an application can call the `nx_arp_info_get` API for statistics on aged entries as well as other useful statistics on ARP table management.
ARP Update Rate – default value 10 seconds	This is the interval between retransmission of ARP queries. When the number of retries reaches the ARP Maximum Retries, NetX abandons the attempt to find a physical mapping of IP address to a MAC address. There is no side effect to reducing this value to a smaller value.

ARP Maximum Queue Depth – default value 4	This is the maximum number of IPv4 packets from the application trying to transmit packets for which NetX has no MAC address mapping. These packets are enqueued while NetX tries to find the MAC address mapping. When the queue is full, the oldest packet is removed. To determine if this is happening, an application can check the `nx_ip_transmit_resource_errors` field in the IP instance data structure using the Expressions view in e^2 studio. Currently there is no API for obtaining this statistic. This value does not need to be changed except under exceptional conditions. It is unusual for the queue to fill up using this default value. If the destination IP address is alive, an ARP response will be received by NetX and corresponding packets on the ARP queue will be transmitted. If the destination IP is dead, it does not make sense to queue any more packets to be sent to that address.
ARP Maximum Retries – default value 18	This is the number of times NetX retransmits an ARP query for an IP address mapping before it gives up. There is no side effect to reducing this value to a smaller value.
ARP Defend by Reply – default value disabled	This is intended for use with hosts who need to keep their current IP address. Normally when a host gets an ARP packet with a matching source IP address but different MAC address, NetX may broadcast an ARP request to **announce** that it owns the IP address. It can only do so if it has not received a conflicting ARP packet within the interval of time specified by the **ARP Defend Interval** property (defined in this section). If it does send a defend reply, it resets the timeout to wait before responding to the next ARP conflict, if one occurs. If this is not the first conflicting ARP packet the host has seen, and the time recorded for the previous conflicting ARP packet is within the **ARP Defend Interval**, then the host MUST immediately cease using this address. This is necessary to ensure that two hosts do not get stuck in an endless loop with both hosts trying to defend the same address. For more details on ARP conflicts, see *RFC 5227 IPv4 Address Conflicts Section 2.4 (b)*. When **ARP Defend by Reply** is defined, an ARP reply is broadcast in addition to the one sent once the **ARP Defend Interval** expires. There is no requirement for a wait interval to expire for this ARP defend packet. This property is used because Windows XP ignores ARP request packets sent out when the **ARP Defend Interval** timeout expires.
ARP Defend Interval – default value 10 seconds	Interval during which a NetX host may not send an ARP defend packet if it receives a conflict ARP packet. The default value recommended is10 seconds in the RFC 5227 to handle IPv4 address conflicts. After NetX sends the ARP defend packet, it resets the timeout back to this value.
ARP Mac Change Notification – default value Disabled	If NetX receives an ARP packet whose MAC address matches an entry in the ARP cache table, and this feature is enabled, NetX will call the ARP collision handler for the application to examine the packet and decide what to do with it. Without this feature, NetX will update the entry in its ARP table to the new MAC address. This behavior, that is a normal ARP protocol, can be taken advantage of in what is called the **man in the middle attack**, ARP cache poisoning or **spoofing**. This enables an attacker to redirect packets from one host to another by altering the MAC address information in the ARP table. NetX has internal handlers for these situations so applications need not handle them.

5.1.5 TCP Properties

For the following group of rate setting properties, NetX timing is based on the NX_IP_PERIODIC_RATE setting. NX_IP_PERIODIC_RATE is derived directly from TX_TIMER_TICKS_PER_SECOND. The latter defaults to 100, but can be optionally be user defined in tx_port.h. If it is defined, NX_IP_PERIODIC_RATE is set to that value. If it is not defined, then NetX defaults it to 100 ticks (10 msec/tick).

TCP Fast Timer Rate – default value 10	This determines the interval on how the fast timer periodic executes in NetX. If set to 10, and NX_IP_PERIODIC_RATE is set to 100, the fast-periodic timer executes every 100 msec. This timer is used to decrement the delayed ACK timeout (see **TCP ACK Timer Rate**) and the socket timeout (see **TCP Transmit Timer Rate**) for when to retransmit the ACK and data packets respectively. Increasing the **TCP Faster Timer Rate** shortens intervals between how the fast timer executes. In fact, it can potentially degrade performance for the extra overhead of processing the fast-periodic timer more often.
TCP Retransmit Timer Rate – default value 1 (1 second)	This value is used to set the TCP socket timeout value. When a TCP socket sends or receives a SYN packet, or sends a data packet, it waits for an acknowledgment (ACK). If the socket timeout expires before receiving an ACK, the timeout is reset (and the packet retransmitted) up to a maximum of the **TCP Maximum Retries**. After that, NetX closes the connection. It is generally not recommended to reduce this value. It will not make the TCP transactions happen quicker.
TCP ACK Timer Rate – default value 5	This determines the interval between what NetX retransmits an ACK for missing data (or ACK probe). The NetX fast periodic timer decrements this timeout on each iteration of the faster timer periodic. If it expires, NetX sends another ACK packet. If a data packet is received, NetX resets the timeout. When data is sent from the NetX, it will also reset the timeout. If it expires, NetX increments the number of retries and resends an ACK. When the maximum number retries is reached (see **TCP Maximum Retries** defined in this section), NetX closes the connection. A setting of 5, with NX_IP_PERIODIC_RATE set to 100, yields a delayed timeout of 200 msec. The greater the NX_TCP_ACK_TIMER_RATE the smaller the ACK timeout. Increasing the value does not increase performance or response time to the **NetX ACK**. It only sends them after a shorter interval.
TCP Maximum Retries – default value 10	When a socket timeout expires without receiving a response from the TCP peer, the socket timeout is reset up until the number of retries equals the **TCP Maximum Retries**.
TCP Retry Shift – default value 0	This is the bit shift applied to the retransmission interval. The default value of zero keeps the interval constant between socket retries (to get a response from the TCP peer). If it is set to 1, it doubles the interval; bit-shifts the timeout value by 1. This value is not often modified.
TCP Maximum TX Queue – default value 20	This is the maximum number of packets that NetX will enqueue on a TCP socket for transmission and retransmission. A packet is enqueued when the socket is waiting to receive an ACK for the data, or it is waiting for the receive window of the other side to increase so it can send the data. For applications using smaller packet pools; limited memory resources, this value can be reduced to prevent packet pool depletion where a significant number of packets are sitting on the transmit queue, unavailable for other packet transmissions. When the TCP socket has reached the maximum number of packets it can enqueue, the TCP socket will send call returns as a NX_TCP_QUEUE_DEPTH error. If it cannot send a packet because the TCP receive window is too small, it returns an NX_WINDOW_OVERFLOW error.

TCP Keepalive – default value disabled	The Keepalive feature starts a timer on the TCP socket in the established state; connected to a peer. When the timer expires, NetX sends a Keepalive ACK to the peer. Receiving a SYN or ACK packet, an ACK packet in response to NetX device's Keep Alive Ack, or any TCP packet with a valid sequence number resets the timer and the retry count. When NetX initiates a Keepalive ACK exchange, it sends a Keepalive ACK packet with the ACK packet's sequence number decreased by one. This is how a TCP peer can distinguish a keepalive ACK from an ACK that indicates 1 byte of data has been sent. When Keepalive is enabled on a socket, NetX also periodically checks if the other side has sent a Keepalive ACK. If the Keepalive timer expires without a response from the peer, the number of Keepalive retries is incremented. If the retries reaches the TCP Keepalive Retries maximum, NetX deems the connection broken and terminates it. Without Keepalive, a TCP connection can persist indefinitely if neither side is transmitting a packet. In that situation, there is no way to know if the socket connection should be closed or remain open.
TCP Keepalive Retries – default value 10	After 10 tries to get a response to a keepalive ACK, NetX resets **closes** the connection.
TCP Keepalive Initial – default value 7200 seconds (2 hours)	The interval before first Keepalive is sent; when a connection completes, or a response is received from a previously sent Keepalive packet.
TCP Keepalive Retry – default value 75 seconds	NetX waits for the time specified by this option before resending another Keepalive packet, unless it has received a response to a previously sent Keepalive packet from the TCP peer.
TCP Window Scaling – default value disabled	This feature allows a TCP receive window to exceed 65k up to a theoretical maximum of 1,073,725,440 bytes. When a NetX TCP Client socket initiates a connection, NetX computes a scaling factor based on the window size (set when the TCP socket is created). The window scaling values are exchanged during the TCP connection establishment phase. Note that the lack of window scaling option in peer's SYN packet is an indication that the peer does not support window scaling. In this situation, window scaling is not used for this connection.
TCP Maximum Out of Order Packets – default value disabled	If enabled, this option sets the maximum number of out-of-order packets stored on the TCP socket receive queue. Subsequent out-of-order packets received are dropped if the socket receive queue has the maximum out of order packets. Eventually the NetX host should send an ACK indicating the missing data to the sender and get the dropped packet retransmitted. At this point, NetX can quickly process all the rest of the data on the receive queue and release the packets. This feature is useful if a packet is lost or dropped, and the sender keeps sending packets. The TCP socket must enqueue all packets back to the missing packet on the TCP socket receive queue. It cannot release any because it is waiting for the missing data to be retransmitted. If the depth of the receive queue is not limited, this can starve the packet pool, rendering the NetX host effectively unresponsive. This can happen in a **bursty** data transmission, where many packets are sent by the TCP peer. A dropped packet may subsequently lead to many packets enqueued on the TCP receive queue.
TCP MSS Minimum – default value 128	MSS, or maximum segment size is the largest amount of TCP data that will not require fragmentation. The **minimum** MSS is the lowest MSS that a NetX TCP socket will accept from a TCP peer. If the MSS parsed from the TCP header option data is below this value, the connection is dropped. The intended usage of this is when an application wants to reject connections with small MSS values.

TCP ACK every N Packets – default is disabled	To enable this feature, enter a positive number. NetX sends an ACK out for every other data packet with new data, not retransmitted data. This is usually the optimal setting to minimize network traffic, packet processing and keep the TCP peer advertise window up to date. Note: if **TCP Immediate ACK** is enabled, this value is overridden; it is set to 1 automatically. Increasing the value increases the possibility the other side must wait on the ***n*th ACK** to know the advertise window has increased enough to send more data. This may result in slower network throughput. The optimal setting is generally based on testing. This ratio of ACKs per N packets is associated with the **delayed ACK** logic in TCP. For applications like Telnet, a higher N can greatly optimize the work on the Telnet Server receiving many 1-byte packets.
TCP Immediate ACK – default value disabled	If enabled, NetX sends an ACK for every packet of new data received. This is useful where a delayed ACK is not desirable.
Reset Disconnect – default value enabled	When enabled, this allows an application to call a disconnect on a TCP socket with a zero-wait option without sending a RST packet. It simply sends a FIN packet to indicate it is initiating a disconnection and closes the socket. It does not wait for an ACK or FIN ACK from the TCP peer. This is useful for hosts that do not want to wait to close a socket, such as servers wishing to free up sockets for the next Client request, and can do so without indicating something is wrong like a RST packet usually does.
RX Size Checking – default value enabled	If enabled, NetX checks that a received packet has at least enough room for the IP or transport layer header; TCP, UDP, IGMP and ICMP depending on where the packet is being processed. For example, in the IP layer NetX checks if the packet has the minimum room needed for the IP header. In the TCP layer, NetX checks if the packet has the minimum room needed for the TCP header. As a packet travels up the stack in NetX, the packet **prepend** pointer and packet length are adjusted for each network layer. When the packet passes from the IP layer to the TCP layer, it moves the prepend pointer to the start of the TCP header, and subtracts the size of the packet length by the size of the IP layer. Similarly, going from the TCP layer to the application layer, the prepend pointer is moved past the TCP header to the application data or header in the case of something like an HTTP packet. The packet length is subtracted by the size of the TCP header. This internal manipulation of the packet eliminates the need for the application adjust pointers and data size. An application or NetX protocol will typically make a socket call such as the `nx_tcp_socket_receive` API. If any packets are waiting on a socket receive queue, the application receives the packet and knows where the data is located and how much there is. The application can use the `_nx_packet_length_get` API (preferably) to obtain the packet data size or to access the `nx_packet_length` field in the `NX_PACKET` instance directly.

5.1.6 NetX and NetX Duo Statistics Properties

If enabled, NetX keeps statistics on its internal operations. These statistics are at the component level, like IP, TCP, ARP and Packet (pool). For TCP and UDP, there are statistics for all TCP/UDP transmissions and statistics per socket. Disabling these statistics reduces processing time slightly.

The value of these statistics is to be able to diagnose problems or optimize network performance without having to stop or interrupt program flow or write tedious debug code.

Examples:

`nx_packet_pool_info_get.c`

If an application does not appear to be sending or transmitting packets, check the `nx_packet_pool_empty_requests` statistic. This is incremented every time `nx_packet_allocate` fails because no packets are in the packet pool. This information is helpful if `nx_packet_allocate` is called from a void function or by a NetX protocol, which may return a different value.

`nx_ip_info_get.c`

If an application is not receiving data, but packets are visible on a third-party packet trace, check the `nx_ip_total_packets_received` statistic to see if the data is forwarded at least as far as the IP level. Similarly, for not seeing packets from the NetX device on a packet trace is to check the `nx_ip_total_packets_sent` statistic.

The following information table is a partial list defining APIs for NetX statistics.

API	NetX statistical information

IP info	Level of the IP layer receiving packets from the driver, as well as forwarding packets from the transport layer to the driver:
	`UINT  `**`_nx_ip_info_get`**`(NX_IP *ip_ptr, ULONG *ip_total_packets_sent,`
	`ULONG *ip_total_bytes_sent, ULONG *ip_total_packets_received,`
	`ULONG *ip_total_bytes_received, ULONG *ip_invalid_packets,`
	`ULONG *ip_receive_packets_dropped, ULONG *ip_receive_checksum_errors,`
	`ULONG *ip_send_packets_dropped, ULONG *ip_total_fragments_sent,`
	`ULONG *ip_total_fragments_received)`
	`nx_ip_interface_info_get.c` is the same but specific to the specified interface.

ARP info	The ARP packets sent and received, and ARP table statistics:
	`UINT  `**`_nx_arp_info_get`**`(NX_IP *ip_ptr, ULONG *arp_requests_sent,`
	`ULONG *arp_requests_received,`
	`ULONG *arp_responses_sent, ULONG *arp_responses_received,`
	`ULONG *arp_dynamic_entries, ULONG *arp_static_entries,`
	`ULONG *arp_aged_entries, ULONG *arp_invalid_messages)`

Packet Pool info	The available packets, empty packet (pool) requests, and invalid packet releases; invalid packet pool or packet pointer supplied.
	`INT  `**`_nx_packet_pool_info_get`**`(NX_PACKET_POOL *pool_ptr, ULONG *total_packets,`
	`ULONG *free_packets,`
	`ULONG *empty_pool_requests, ULONG *empty_pool_suspensions,`
	`ULONG *invalid_packet_releases)`

API	NetX statistical information

TCP info	Total number of TCP packets sent/received, number of connections and disconnections, and dropped and retransmitted packets. The socket-specific API includes the receive window size.
	`UINT _nx_tcp_info_get(NX_IP *ip_ptr, ULONG *tcp_packets_sent, ULONG`
	`*tcp_bytes_sent,`
	`ULONG *tcp_packets_received, ULONG`
	`*tcp_bytes_received,`
	`ULONG *tcp_invalid_packets,`
	`ULONG *tcp_receive_packets_dropped,`
	`ULONG *tcp_checksum_errors, ULONG`
	`*tcp_connections,`
	`ULONG *tcp_disconnections, ULONG`
	`*tcp_connections_dropped,`
	`ULONG *tcp_retransmit_packets)`
	`UINT _nx_tcp_socket_info_get(NX_TCP_SOCKET *socket_ptr, ULONG`
	`*tcp_packets_sent,`
	`ULONG *tcp_bytes_sent,`
	`ULONG *tcp_packets_received, ULONG`
	`*tcp_bytes_received,`
	`ULONG *tcp_retransmit_packets, ULONG`
	`*tcp_packets_queued,`
	`ULONG *tcp_checksum_errors, ULONG *tcp_socket_state,`
	`ULONG *tcp_transmit_queue_depth,`
	`ULONG *tcp_transmit_window,`
	`ULONG *tcp_receive_window)`

UDP info	Total number of UDP packets sent/received, dropped packets, and improperly formed packets to send or receive. The socket-specific API does not include the count of invalid packets received.
	`UINT` **`_nx_udp_info_get`** (NX_IP *ip_ptr, ULONG *udp_packets_sent, ULONG
	*udp_bytes_sent,
	`ULONG *udp_packets_received, ULONG`
	`*udp_bytes_received,`
	`ULONG *udp_invalid_packets,`
	`ULONG *udp_receive_packets_dropped,`
	`ULONG *udp_checksum_errors)`
	`UINT` **`_nx_udp_socket_info_get`** (NX_UDP_SOCKET *socket_ptr, ULONG
	*udp_packets_sent,
	`ULONG *udp_bytes_sent, ULONG *udp_packets_received,`
	`ULONG *udp_bytes_received, ULONG *udp_packets_queued,`
	`ULONG *udp_receive_packets_dropped,`
	`ULONG *udp_checksum_errors)`

ICMP info	Control message transmission, including count of unsupported ICMP messages, ping requests that timed out, and responses to ping request.
	`UINT _nx_icmp_info_get(NX_IP *ip_ptr, ULONG *pings_sent, ULONG`
	`*ping_timeouts,`
	`ULONG *ping_threads_suspended,`
	`ULONG *ping_responses_received,`
	`ULONG *icmp_checksum_errors,`
	`ULONG *icmp_unhandled_messages)`

IGMP info	The multicast groups joined, IGMP queries received, and IGMP reports sent.
	`UINT _nx_igmp_info_get(NX_IP *ip_ptr, ULONG *igmp_reports_sent,`
	`ULONG *igmp_queries_received,`
	`ULONG *igmp_checksum_errors, ULONG`
	`*current_groups_joined)`

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.08.18	—	Initial Release
1.01	Jan.07.19	—	Updated Packet Header Pad Size description and revamped document layout for easier reference.
1.02	Apr.30.19	—	Updated ARP cache size to 520 bytes for SSP 1.6.0

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/.