# Chapter 2 - Installation and Use of NetX Duo

## Contents



## Chapter 2 - Installation and Use of NetX Duo

This chapter contains a description of various issues related to installation, setup, and use of the high- performance network stack NetX Duo, including the following:

### Host Considerations

Embedded development is usually performed on Windows or Linux (Unix) host computers. After the application is compiled, linked, and the executable is generated on the host, it is downloaded to the target hardware for execution.

Usually the target download is done from within the development tool's debugger. After download, the debugger is responsible for providing target execution control (go, halt, breakpoint, etc.) as well as access to memory and processor registers.

Most development tool debuggers communicate with the target hardware via on-chip debug (OCD) connections such as JTAG (IEEE 1149.1) and Background Debug Mode (BDM). Debuggers also communicate with target hardware through In-Circuit Emulation (ICE) connections. Both OCD and ICE connections provide robust solutions with minimal intrusion on the target resident software.

As for resources used on the host, the source code for NetX Duo is delivered in ASCII format and requires approximately 1 Mbytes of space on the host computer's hard disk.

## Target Considerations

NetX Duo requires between 5 KBytes and 45 KBytes of Read-Only Memory (ROM) on the target. Another 1 to 5KBytes of the target's Random Access Memory (RAM) are required for the NetX Duo thread stack and other global data structures.

In addition, NetX Duo requires the use of two ThreadX timer objects and one ThreadX mutex object. These facilities are used for periodic processing needs and thread protection inside the NetX Duo protocol stack.

## Product Distribution

NetX Duo can be obtained from our public source code repository at https://github.com/eclipse-threadx/netxduo/.

The following is a list of several important files in the repository:

**nx_api.h**
C header file containing all system equates, data structures, and service prototypes.

**nx_port.h** C header file containing all development-tool and targetspecific data definitions and structures.

**demo_netx.c** C file containing a small demo application.

**nx.a (or nx.lib)**
Binary version of the NetX Duo C library that is distributed with the standard package.

## NetX Duo Installation

NetX Duo is installed by cloning the GitHub repository to your local machine. The following is typical syntax for creating a clone of the NetX Duo repository on your PC:

```
git clone https://github.com/eclipse-threadx/netxduo
```

Alternatively you can download a copy of the repository using the download button on the GitHub main page.

You will also find instructions for building the NetX Duo library on the front page of the online repository.

> **Important:** *Application software needs access to the NetX Duo library file (usually **nx.a** or **nx.lib**) and the C include files **nx_api.h, and nx_port.h**. This is accomplished either by setting the appropriate path for the development tools or by copying these files into the application development area.*

## Using NetX Duo

Using NetX Duo is easy. Basically, the application code must include ***nx_api.h*** during compilation and link with the NetX Duo library ***nx.a*** (or ***nx.lib***).

The following are the four easy steps required to build a NetX Duo application:

| Step | Description |
|---|---|
| Step 1: | Include the ***nx_api.h*** file in all application files that use NetX Duo services or data structures. |
| Step 2: | Initialize the NetX Duo system by calling ***nx_system_initialize*** from the ***tx_application_define*** function or an application thread. |
| Step 3: | Create an IP instance, enable the Address Resolution Protocol (ARP), if necessary, and any sockets after ***nx_system_initialize*** is called. |
| Step 4: | Compile application source and link with the NetX Duo runtime library ***nx.a*** (or ***nx.lib***). The resulting image can be downloaded to the target and executed! |

## Troubleshooting

Each NetX Duo port is delivered with one or more demonstrations that execute on an actual network or via a simulated network driver. It is always a good idea to get the demonstration system running first.

If the demonstration system does not run properly, perform the following operations to narrow the problem:

1. Determine how much of the demonstration is running.
2. Increase stack sizes in any new application threads.
3. Recompile the NetX Duo library with the appropriate debug options listed in the configuration option section.
4. Examine the NX_IP structure to see if packets are being sent or received.
5. Examine the default packet pool to see if there are available packets.
6. Ensure the network driver is supplying ARP and IP packets with its headers on 4-byte boundaries for applications requiring IPv4 or IPv6 connectivity.
7. Temporarily bypass any recent changes to see if the problem disappears or changes.

## Configuration Options

There are several configuration options when building the NetX Duo library and the application using NetX Duo. The configuration options can be defined in the application source, on the command line, or within the ***nx_user.h*** include file, unless otherwise specified.

> **Note:** *Options defined in* ***nx_user.h*** *are applied only if the application and NetX Duo library are built with* **NX_INCLUDE_USER_DEFINE_FILE** *defined.*

The following sections list the configuration options available in NetX Duo. General options applicable to both IPv4 and IPv6 are listed first, followed by IPv6-specific options.

### System Configuration Options

| Option | Description |
|---|---|
| NX_ASSERT_FAIL | Symbol that defines the debug statement to use when an assertion fails. |
| NX_DEBUG | Defined, enables the optional print debug information available from the RAM Ethernet network driver. |
| NX_DEBUG_PACKET | Defined, enables the optional debug packet dumping available in the RAM Ethernet network driver. |
| NX_DISABLE_ASSERT | Defined, disables ASSERT checks in the source code. By default this option is not defined. |

| Option | Description |
| --- | --- |
| NX_DISABLE_ERROR_CHECKING | Defined, removes the basic NetX Duo error checking API and improves performance. API return codes not affected by disabling error checking are listed in bold typeface in the API definition. This define is typically used after the application is debugged sufficiently and its use improves performance and decreases code size. |
| NX_DRIVER_DEFERRED_PROCESSING | Defined, enables deferred network driver packet handling. This allows the network driver to place a packet on the IP instance and have the real processing routine called from the NetX Duo internal IP helper thread. |
| NX_DUAL_PACKET_POOL_ENABLE | Renamed to ***NX_ENABLE_DUAL_PACKET_POOL***. Although it is still being supported, new designs are encouraged to use ***NX_ENABLE_DUAL_PACKET_POOL***. |
| NX_ENABLE_DUAL_PACKET_POOL | Defined, allows the stack to use two packet pools, one with large payload size and one with smaller payload size. By default this option is not enabled. |
| NX_ENABLE_EXTENDED_NOTIFY_SUPPORT | Defined, enables more callback hooks in the stack. These callback functions are used by the BSD wrapper layer. By default this option is not defined. |
| NX_ENABLE_INTERFACE_CAPABILITY | Defined, allows the interface device driver to specify extra capability information, such as checksum off-loading. By default this option is not defined. |
| NX_ENABLE_SOURCE_ADDRESS_CHECK | Defined, enables the source address of incoming packet to be checked. By default this option is disabled. |
| NX_IPSEC_ENABLE | Defined, enables the NetX Duo library to support IPsec operations. This feature requires the optional NetX Duo IPsec module. By default this feature is not enabled. |

| Option | Description |
|---|---|
| NX_LITTLE_ENDIAN | Defined, performs the necessary byte swapping on little endian environments to ensure the protocol headers are in proper big endian format. Note the default is typically setup in ***nx_port.h***. |
| NX_MAX_PHYSICAL_INTERFACES | Specifies the total number of physical network interfaces on the device. The default value is 1 and is defined in ***nx_api.h***; a device must have at least one physical interface. Note this does not include the loopback interface. |
| NX_NAT_ENABLE | Defined, NetX Duo is built with NAT process. By default this option is not defined. |
| NX_PHYSICAL_HEADER | Specifies the size in bytes of the physical header of the frame. The default value is 16 (based on a typical 14-byte Ethernet frame aligned to 32-bit boundary) and is defined in ***nx_api.h***. The application can override the default by defining the value before ***nx_api.h*** is included, such as in ***nx_user.h***. |
| NX_PHYSICAL_TRAILER | Specifies the size in bytes of the physical packet trailer and is typically used to reserve storage for things like Ethernet CRCs, etc. The default value is 4 and is defined in ***nx_api.h***. |

**ARP Configuration Options**

| Option | Description |
|---|---|
| NX_ARP_DEFEND_BY_REPLY | Defined, allows NetX Duo to defend its IP address by sending an ARP response. |
| NX_ARP_DEFEND_INTERVAL | Defines the interval, in seconds, the ARP module sends out the next defend packet in response to an incoming ARP message that indicates an address in conflict. |

| Option | Description |
| --- | --- |
| NX_ARP_DISABLE_AUTO_ARP_ENTRY | Renamed to ***NX_DISABLE_ARP_AUTO_ENTRY***. Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_ARP_AUTO_ENTRY***. |
| NX_ARP_EXPIRATION_RATE | Specifies the number of seconds ARP entries remain valid. The default value of zero disables expiration or aging of ARP entries and is defined in ***nx_api.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_ARP_MAC_CHANGE_NOTIFICATION_ENABLE | Renamed to ***NX_ENABLE_ARP_MAC_CHANGE_NOTIFICATION***. Although it is still being supported, new designs are encouraged to use ***NX_ENABLE_ARP_MAC_CHANGE_NOTIFICATION***. |
| NX_ARP_MAX_QUEUE_DEPTH | Specifies the maximum number of packets that can be queued while waiting for an ARP response. The default value is 4 and is defined in ***nx_api.h***. |
| NX_ARP_MAXIMUM_RETRIES | Specifies the maximum number of ARP retries made without an ARP response. The default value is 18 and is defined in ***nx_api.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_ARP_UPDATE_RATE | Specifies the number of seconds between ARP retries. The default value is 10, which represents 10 seconds, and is defined in ***nx_api.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_DISABLE_ARP_AUTO_ENTRY | Defined, disables entering ARP request information in the ARP cache. |
| NX_DISABLE_ARP_INFO | Defined, disables ARP information gathering. |
| NX_ENABLE_ARP_MAC_CHANGE_NOTIFICATION | Defined, allows ARP to invoke a callback notify function on detecting the MAC address is updated. |

**ICMP Configuration Options**

| Option | Description |
|---|---|
| NX_DISABLE_ICMP_INFO | Defined, disables ICMP information gathering. |
| NX_DISABLE_ICMP_RX_CHECKSUM | Defined, disables both ICMPv4 and ICMPv6 checksum computation on received ICMP packets. This option is useful when the network interface driver is able to verify the ICMPv4 and ICMPv6 checksum, and the application does not use the IP fragmentation feature or the IPsec feature. By default this option is not defined. |
| NX_DISABLE_ICMP_TX_CHECKSUM | Defined, disables both ICMPv4 and ICMPv6 checksum computation on transmitted ICMP packets. This option is useful where the network interface driver is able to compute the ICMPv4 and ICMPv6 checksum, and the application does not use the IP fragmentation feature or IPsec feature. By default this option is not defined. |
| NX_DISABLE_ICMPV4_ERROR_MESSAGE | Defined, NetX Duo does not send ICMPv4 Error Messages in response to error conditions such as improperly formatted IPv4 header. By default this option is not defined. |
| NX_DISABLE_ICMPV4_RX_CHECKSUM | Defined, disables ICMPv4 checksum computation on received ICMP packets. This option is defined automatically if ***NX_DISABLE_ICMP_RX_CHECKSUM*** is defined. By default this option is not defined. |
| NX_DISABLE_ICMPv4_RX_CHECKSUM | Renamed to ***NX_DISABLE_ICMPV4_RX_CHECKSUM***. Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_ICMPV4_RX_CHECKSUM***. |

| Option | Description |
|---|---|
| NX_DISABLE_ICMPV4_TX_CHECKSUM | Defined, disables ICMPv4 checksum computation on transmitted ICMP packets. This option is defined automatically if ***NX_DISABLE_ICMP_TX_CHECKSUM*** is defined. By default this option is not defined. |
| NX_DISABLE_ICMPv4_TX_CHECKSUM | Renamed to ***NX_DISABLE_ICMPV4_TX_CHECKSUM***.Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_ICMPV4_TX_CHECKSUM***. |
| NX_ENABLE_ICMP_ADDRESS_CHECK | Defined, the destination address of ICMP packet is checked. The default is disabled. An ICMP Echo Request destined to an IP broadcast or IP multicast address will be silently discarded. |

**IGMP Configuration Options**

| Option | Description |
|---|---|
| NX_DISABLE_IGMP_INFO | Defined, disables IGMP information gathering. |
| NX_DISABLE_IGMPV2 | Defined, disables IGMPv2 support, and NetX Duo supports IGMPv1 only. By default this option is not set and is defined in ***nx_api.h***. |
| NX_MAX_MULTICAST_GROUPS | Specifies the maximum number of multicast groups that can be joined. The default value is 7 and is defined in ***nx_api.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |

**IP Configuration Options**

| Option | Description |
|---|---|
| NX_DISABLE_FRAGMENTATION | Defined, disables both IPv4 and IPv6 fragmentation and reassembly logic. |

| Option | Description |
|---|---|
| NX_DISABLE_IPV4 | Defined, disables IPv4 functionality. This option can be used to build NetX Duo to support IPv6 only. By default this option is not defined. |
| NX_DISABLE_IP_INFO | Defined, disables IP information gathering. |
| NX_DISABLE_IP_RX_CHECKSUM | Defined, disables checksum logic on received IPv4 packets. This is useful if the network device is able to verify the IPv4 checksum, and the application does not expect to use IP fragmentation or IPsec. |
| NX_DISABLE_IP_TX_CHECKSUM | Defined, disables checksum logic on IPv4 packets sent. This is useful in situations in which the underlying network device is capable of generating the IPv4 header checksum, and the application does not expect to use IP fragmentation or IPsec. |
| NX_DISABLE_LOOPBACK_INTERFACE | Defined, disables NetX Duo support for the loopback interface. |
| NX_DISABLE_RX_SIZE_CHECKING | Defined, disables the size checking on received packets. |
| NX_ENABLE_IP_RAW_PACKET_FILTER | Defined, enables the IP raw packet receive filter functionality. Applications requiring more control over the type of raw IP packets to be received can use this feature. The IP raw packet filter feature also supports the raw socket operation in the BSD compatibility layer. By default this option is not defined. |
| NX_ENABLE_IP_STATIC_ROUTING | Defined, enables IPv4 static routing in which a destination address can be assigned a specific next hop address. By default IPv4 static routing is disabled. |
| NX_FRAGMENT_IMMEDIATE_ASSEMBLY | Defined, allows IPv4 and IPv6 reassembly logic to execute right away after receiving an IP fragment. By default this option is not defined. |

| Option | Description |
|---|---|
| NX_IP_MAX_REASSEMBLY_TIME | Symbol that controls maximum time allowed to reassemble IPv4 fragment and IPv6 fragment. Note the value defined here overwrites both ***NX_IPV4_MAX_REASSEMBLY_TIME*** and ***NX_IPV6_MAX_REASSEMBLY_TIME***. |
| NX_IP_PERIODIC_RATE | Defined, specifies the number of ThreadX timer ticks in one second. The default value is derived from the ThreadX symbol ***TX_TIMER_TICKS_PER_SECOND,*** which by default is set to 100 (10ms timer). Applications shall exercise caution when modifying this value, as the rest of the NetX Duo modules derive timing information from ***NX_IP_PERIODIC_RATE.*** |
| NX_IP_RAW_MAX_QUEUE_DEPTH | Symbol that controls the number of raw IP packets can be queued on the raw packet receive queue. By default value is set to 20. |
| NX_IP_ROUTING_TABLE_SIZE | Defined, sets the maximum number of entries in the IPv4 static routing table, which is a list of an outgoing interface and the next hop addresses for a given destination address. The default value is 8 and is defined in ***nx_api.h.*** This symbol is used only if ***NX_ENABLE_IP_STATIC_ROUTING*** is defined. |
| NX_IPV4_MAX_REASSEMBLY_TIME | Symbol that controls maximum time allowed to reassemble IPv4 fragment. Note the value defined in NX_IP_MAX_REASSEMBLY_TIME overwrites this value. |
| NX_ENABLE_TCPIP_OFFLOAD | Symbol that enables TCP/IP offload feature. Note NX_ENABLE_INTERFACE_CAPABILITY must be defined to enable this feature. |

**Packet Configuration Options**

| Option | Description |
|---|---|
| NX_DISABLE_PACKET_CHAIN | Defined, disables the packet chain logic. By default this is not defined. |
| NX_DISABLE_PACKET_INFO | Defined, disables packet pool information gathering. |
| NX_ENABLE_LOW_WATERMARK | Defined, enables NetX Duo packet pool low watermark feature. Application sets low watermark value. On receiving TCP packets, if the packet pool low watermark is reached, NetX Duo silently discards the packet by releasing it, preventing the packet pool from starvation. By default this feature is not enabled. |
| NX_ENABLE_PACKET_DEBUG_INFO | Defined, logs packet debug information. |
| NX_PACKET_ALIGNMENT | Defined, specifies the alignment requirement, in bytes, for starting address of the packet payload area. This option deprecates ***NX_PACKET_HEADER_PAD*** and ***NX_PACKET_HEADER_PAD_SIZE***. By default this option is defined to be 4, making the starting address of the payload area 4-byte aligned. |
| NX_PACKET_HEADER_PAD | Defined, enables padding towards the end of the NX_PACKET control block. The number of ULONG words to pad is defined by ***NX_PACKET_HEADER_PAD_SIZE***. Note this option is depreciated by ***NX_PACKET_ALIGNMENT***. |

| Option | Description |
|---|---|
| NX_PACKET_HEADER_PAD_SIZE | Sets the number of ULONG words to be padded to the NX_PACKET structure, allowing the packet payload area to start at the desired alignment. This feature is useful when receive buffer descriptors point directly into NX_PACKET payload area, and the network interface receive logic or the cache operation logic expects the buffer starting address to meet certain alignment requirements. This value becomes valid only when ***NX_PACKET_HEADER_PAD*** is defined. Note this option is deprecated by ***NX_PACKET_ALIGNMENT***. |

**RARP Configuration Options**

| Option | Description |
|---|---|
| NX_DISABLE_RARP_INFO | Defined, disables RARP information gathering. |

**TCP Configuration Options**

| Option | Description |
|---|---|
| NX_DISABLE_RESET_DISCONNECT | Defined, disables the reset processing during disconnect when the timeout value supplied is specified as ***NX_NO_WAIT***. |
| NX_DISABLE_TCP_INFO | Defined, disables TCP information gathering. |
| NX_DISABLE_TCP_RX_CHECKSUM | Defined, disables checksum logic on received TCP packets. This is only useful in situations in which the link-layer has reliable checksum or CRC processing, or the interface driver is able to verify the TCP checksum in hardware, and the application does not use IPsec. |

| Option | Description |
|---|---|
| NX_DISABLE_TCP_TX_CHECKSUM | Defined, disables checksum logic for sending TCP packets. This is only useful in situations in which the receiving network node has received TCP checksum logic disabled or the underlying network driver is capable of generating the TCP checksum, and the application does not use IPsec. |
| NX_ENABLE_TCP_KEEPALIVE | Defined, enables the optional TCP keepalive timer. The default settings is not enabled. |
| NX_ENABLE_TCP_MSS_CHECK | Defined, enables the verification of minimum peer MSS before accepting a TCP connection. To use this feature, the symbol ***NX_ENABLE_TCP_MSS_MINIMUM*** must be defined. By default, this option is not enabled. |
| NX_ENABLE_TCP_QUEUE_DEPTH_UPDATE_NOTIFY | Defined, allows the application to install a callback function that is invoked when the TCP transmit queue depth is no longer at maximum value. This callback serves as an indication that the TCP socket is ready to transmit more data. By default this option is not enabled. |
| NX_ENABLE_TCP_WINDOW_SCALING | Enables the window scaling option for TCP applications. If defined, window scaling option is negotiated during TCP connection phase, and the application is able to specify a window size larger than 64K. The default setting is not enabled (not defined). |
| NX_MAX_LISTEN_REQUESTS | Specifies the maximum number of server listen requests. The default value is 10 and is defined in ***nx_api.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |

| Option | Description |
|---|---|
| NX_TCP_ACK_EVERY_N_PACKETS | Specifies the number of TCP packets to receive before sending an ACK. Note if ***NX_TCP_IMMEDIATE_ACK*** is enabled but ***NX_TCP_ACK_EVERY_N_PACKETS*** is not, this value is automatically set to 1 for backward compatibility. |
| NX_TCP_ACK_TIMER_RATE | Specifies how the number of system ticks (NX_IP_PERIODIC_RATE) is divided to calculate the timer rate for the TCP delayed ACK processing. The default value is 5, which represents 200ms, and is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_TCP_ENABLE_KEEPALIVE | Renamed to ***NX_ENABLE_TCP_KEEPALIVE***. Although it is still being supported, new designs are encouraged to use ***NX_ENABLE_TCP_KEEPALIVE***. |
| NX_TCP_ENABLE_MSS_CHECK | Renamed to ***NX_ENABLE_TCP_MSS_CHECK***.Although it is still being supported, new designs are encouraged to use ***NX_ENABLE_TCP_MSS_CHECK.*** |
| NX_TCP_ENABLE_WINDOW_SCALING | Renamed to ***NX_ENABLE_TCP_WINDOW_SCALING***.Although it is still being supported, new designs are encouraged to use ***NX_ENABLE_TCP_WINDOW_SCALING***. |

| Option | Description |
|---|---|
| NX_TCP_FAST_TIMER_RATE | Specifies how the number of NetX Duo internal ticks (NX_IP_PERIODIC_RATE) is divided to calculate the fast TCP timer rate. The fast TCP timer is used to drive the various TCP timers, including the delayed ACK timer. The default value is 10, which represents 100ms assuming the ThreadX timer is running at 10ms. This value is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_TCP_IMMEDIATE_ACK | Defined, enables the optional TCP immediate ACK response processing. Defining this symbol is equivalent to defining ***NX_TCP_ACK_EVERY_N_PACKETS*** to be 1. |
| NX_TCP_KEEPALIVE_INITIAL | Specifies the number of seconds of inactivity before the keepalive timer activates. The default value is 7200, which represents 2 hours, and is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_TCP_KEEPALIVE_RETRIES | Specifies how many keepalive retries are allowed before the connection is deemed broken. The default value is 10, which represents 10 retries, and is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_TCP_KEEPALIVE_RETRY | Specifies the number of seconds between retries of the keepalive timer assuming the other side of the connection is not responding. The default value is 75, which represents 75 seconds between retries, and is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |

| Option | Description |
| --- | --- |
| NX_TCP_MAX_OUT_OF_ORDER_PACKETS | Symbol that defines the maximum number of out-of-order TCP packets can be kept in the TCP socket receive queue. This symbol can be used to limit the number of packets queued in the TCP receive socket, preventing the packet pool from being starved. By default this symbol is not defined, thus there is no limit on the number of out of order packets being queued in the TCP socket. |
| NX_TCP_MAXIMUM_RETRIES | Specifies how many data transmit retries are allowed before the connection is deemed broken. The default value is 10, which represents 10 retries, and is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_TCP_MAXIMUM_RX_QUEUE | Symbol that defines the maximum receive queue for TCP sockets. This feature is enabled by ***NX_ENABLE_LOW_WATERMARK***. |
| NX_TCP_MAXIMUM_TX_QUEUE | Specifies the maximum depth of the TCP transmit queue before TCP send requests are suspended or rejected. The default value is 20, which means that a maximum of 20 packets can be in the transmit queue at any given time. Note packets stay in the transmit queue until an ACK that covers some or all of the packet data is received from the other side of the connection. This constant is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_TCP_MSS_MINIMUM | Symbol that defines the minimal MSS value NetX Duo TCP module accepts. This feature is enabled by ***NX_ENABLE_TCP_MSS_CHECK***. |

| Option | Description |
| --- | --- |
| NX_TCP_QUEUE_DEPTH_UPDATE_NOTIFY | Retained to ENABLE ***NX_ENABLE_TCP_QUEUE_DEPTH_UPDATE_NOTIF*** Although it is still being supported, new designs are encouraged to use ***NX_ENABLE_TCP_QUEUE_DEPTH_UPDATE_NOTIF*** |
| NX_TCP_RETRY_SHIFT | Specifies how the retransmit timeout period changes between retries. If this value is 0, the initial retransmit timeout is the same as subsequent retransmit timeouts. If this value is 1, each successive retransmit is twice as long. If this value is 2, each subsequent retransmit timeout is four times as long. The default value is 0 and is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |
| NX_TCP_TRANSMIT_TIMER_RATE | Specifies how the number of system ticks (***NX_IP_PERIODIC_RATE***) is divided to calculate the timer rate for the TCP transmit retry processing. The default value is 1, which represents 1 second, and is defined in ***nx_tcp.h***. The application can override the default by defining the value before ***nx_api.h*** is included. |

**UDP Configuration Options**

| Option | Description |
| --- | --- |
| NX_DISABLE_UDP_INFO | Defined, disables UDP information gathering. |
| NX_DISABLE_UDP_RX_CHECKSUM | Defined, disables the UDP checksum computation on incoming UDP packets. This is useful if the network interface driver is able to verify UDP header checksum in hardware, and the application does not enable IPsec or IP fragmentation logic. |

| Option | Description |
| --- | --- |
| NX_DISABLE_UDP_TX_CHECKSUM | Defined, disables the UDP checksum computation on outgoing UDP packets. This is useful if the network interface driver is able to compute UDP header checksum and insert the value in the IP head before transmitting the data, and the application does not enable IPsec or IP fragmentation logic. |

**IPv6 Options**

| Option | Description |
| --- | --- |
| NX_DISABLE_IPV6 | Disables IPv6 functionality when the NetX Duo library is built. For applications that do not need IPv6, this avoids pulling in code and additional storage space needed to support IPv6. |
| NX_DISABLE_IPV6_PATH_MTU_DISCOVERY | Defined, disables path MTU discovery, which is used to determine the maximum MTU in the path to a target in the NetX Duo host destination table. This enables the NetX Duo host to send the largest possible packet that will not require fragmentation. By default, this option is defined (path MTU is disabled). |
| NX_ENABLE_IPV6_ADDRESS_CHANGE_NOTIFY | Defined, allows a callback function to be invoked when the IPv6 address is changed. By default this option is not enabled. |
| NX_ENABLE_IPV6_MULTICAST | Defined, enables IPv6 multicast join/leave function. By default this option is not enabled. |
| NX_ENABLE_IPV6_PATH_MTU_DISCOVERY | Defined, enables the IPv6 path MTU discovery feature. By default this option is not enabled. |

| Option | Description |
| --- | --- |
| NX_IPV6_ADDRESS_CHANGE_NOTIFY_ENABLE | Renamed to ***NX_ENABLE_IPV6_ADDRESS_CHANGE_NOTIFY***. Although it is still being supported, new designs are encouraged to use ***NX_ENABLE_IPV6_ADDRESS_CHANGE_NOTIFY***. |
| NX_IPV6_DEFAULT_ROUTER_TABLE_SIZE | Specifies the number of entries in the IPv6 routing table. At least onS entry is needed for the default router. Defined in ***nx_api.h***, the default value is 8. |
| NX_IPV6_DESTINATION_TABLE_SIZE | Specifies the number of entries in the IPv6 destination table. This stores information about next hop addresses for IPv6 addresses. Defined in ***nx_api.h***, the default value is 8. |
| NX_IPV6_MAX_REASSEMBLY_TIME | Symbol that controls the maximum time allowed to reassemble IPv6 fragment. |
| NX_IPV6_MULTICAST_ENABLE | Renamed to ***NX_ENABLE_IPV6_MULTICAST***. Although it is still being supported, new designs are encouraged to use ***NX_ENABLE_IPV6_MULTICAST***. |
| NX_IPV6_PREFIX_LIST_TABLE_SIZE | Specifies the size of the prefix table. Prefix information is obtained from router advertisements and is part of the IPv6 address configuration. Defined in ***nx_api.h***, the default value is 8. |
| NX_IPV6_STATELESS_AUTOCONFIG_CONTROL | Defined, allows NetX Duo to disable stateless address autoconfiguration feature. By default this option is not enabled. |
| NX_MAX_IPV6_ADDRESSES | Specifies the number of entries in the IPv6 address pool. During interface configuration, NetX Duo uses IPv6 entries from the pool. It is defaulted to (NX_MAX_PHYSICAL_INTERFACES * 3) to allow each interface to have at least one link local address and two global addresses. Note that all interfaces share the IPv6 address pool. |

| Option | Description |
| --- | --- |
| NX_PATH_MTU_INCREASE_WAIT_INTERVAL | Specifies the wait interval in timer ticks to reset the path MTU for a specific target in the destination table. If ***NX_DISABLE_IPV6_PATH_MTU_DISCOVERY*** is defined, defining this symbol has no effect. |
| NX_PATH_MTU_INCREASE_WAIT_SYMBOL_INTERVAL | This symbol specifies the wait interval (in seconds) to reset the path MTU value for a destination table entry. It is valid only if ***NX_ENABLE_IPV6_PATH_MTU_DISCOVERY*** is defined. By default this value is set to 600 (seconds). |

**Neighbor Cache Configuration Options**

| Option | Description |
| --- | --- |
| NX_DELAY_FIRST_PROBE_TIME | Specifies the delay in seconds before the first solicitation is sent out for a cache entry in the STALE state. Defined in ***nx_nd_cache.h***, the default value is 5. |
| NX_DISABLE_IPV6_DAD | Defined, this option disables Duplicate Address Detection (DAD) during IPv6 address assignment. Addresses are set either by manual configuration or through Stateless Address Auto Configuration. |
| NX_DISABLE_IPV6_PURGE_UNUSED_CACHE_ENTRIES | Defined, this option prevents NetX Duo from removing older cache table entries before their timeout expires to make room for new entries when the table is full. Static and router entries are never purged. |

| Option | Description |
| --- | --- |
| NX_IPV6_DAD_TRANSMITS | Specifies the number of Neighbor Solicitation messages to be sent before NetX Duo marks an interface address as valid. If ***NX_DISABLE_IPV6_DAD*** is defined (DAD disabled), setting this option has no effect. Alternatively, a value of zero (0) turns off DAD but leaves the DAD functionality in NetX Duo. Defined in ***nx_api.h***, the default value is 3. |
| NX_IPV6_DISABLE_PURGE_UNUSED_CACHE_ENTRIES | Defined to ***NX_DISABLE_IPV6_PURGE_UNUSED_CACHE_ENTI*** Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_IPV6_PURGE_UNUSED_CACHE_ENTI*** |
| NX_IPV6_NEIGHBOR_CACHE_SIZE | Specifies the number of entries in the IPv6 Neighbor Cache table. Defined in ***nx_nd_cache.h***, the default value is 16. |
| NX_MAX_MULTICAST_SOLICIT | Specifies the number of Neighbor Solicitation messages NetX Duo transmits as part of the IPv6 Neighbor Discovery protocol when mapping between IPv6 address and MAC address is required. Defined in ***nx_nd_cache.h***, the default value is 3. |
| NX_MAX_UNICAST_SOLICIT | Specifies the number of Neighbor Solicitation messages NetX Duo transmits to determine a specific neighbor's reachability. Defined in ***nx_nd_cache.h***, the default value is 3. |
| NX_ND_MAX_QUEUE_DEPTH | Symbol that defines the maximum number of packets queued up for ND cache to be resolved. By default this symbol is set to 4. |

| Option | Description |
|---|---|
| NX_REACHABLE_TIME | Specifies the time out in seconds for a cache entry to exist in the REACHABLE state with no packets received from the cache destination IPv6 address. Defined in ***nx_nd_cache.h***, the default value is 30. |
| NX_RETRANS_TIMER | Specifies in milliseconds the length of delay between solicitation packets sent by NetX Duo. Defined in ***nx_nd_cache.h***, the default value is 1000. |
| NXDUO_DISABLE_DAD | Renamed to ***NX_DISABLE_IPV6_DAD***. Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_IPV6_DAD***. |
| NXDUO_DUP_ADDR_DETECT_TRANSMITS | Renamed to ***NX_IPV6_DAD_TRANSMITS***. Although it is still being supported, new designs are encouraged to use ***NX_IPV6_DAD_TRANSMITS***. |

**Miscellaneous ICMPv6 Configuration Options**

| Option | Description |
|---|---|
| NX_DISABLE_ICMPV6_ERROR_MESSAGE | Defined, disables NetX Duo from sending an ICMPv6 error message in response to a problem packet (e.g., improperly formatted header or packet header type is deprecated) received from another host. |
| NX_DISABLE_ICMPV6_REDIRECT_PROCESS | Defined, disables ICMPv6 redirect packet processing. NetX Duo by default processes redirect messages and updates the destination table with next hop IP address information. |
| NX_DISABLE_ICMPV6_ROUTER_ADVERTISEMENT_PROCESS | Defined, disables NetX Duo from processing information received in IPv6 router advertisement packets. |

| Option | Description |
| --- | --- |
| NX_DISABLE_ICMPV6_ROUTER_SOLICITATION | Defined, disables NetX Duo from sending IPv6 router solicitation messages at regular intervals to the router. |
| NX_DISABLE_ICMPV6_RX_CHECKSUM | Defined, disables ICMPv6 checksum computation on received ICMP packets. |
| NX_DISABLE_ICMPv6_RX_CHECKSUM | Renamed to ***NX_DISABLE_ICMPV6_RX_CHECKSUM***. Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_CMPV6_RX_CHECKSUM***. |
| NX_DISABLE_ICMPV6_TX_CHECKSUM | Defined, disables and ICMPv6 checksum computation on transmitted ICMP packets. |
| NX_DISABLE_ICMPV6_TX_CHECKSUM | Renamed to ***NX_DISABLE_ICMPV6_TX_CHECKSUM***. Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_ICMPV6_TX_CHECKSUM***. |
| NX_ICMPV6_MAX_RTR_SOLICITATIONS | Define the max number of router solicitations a host sends until a router response is received. If no response is received, the host concludes no router is present. The default value is 3. |
| NX_ICMPV6_RTR_SOLICITATION_DELAY | Specifies the maximum delay for the initial router solicitation in seconds. |
| NX_ICMPV6_RTR_SOLICITATION_INTERVAL | Specifies the interval between two router solicitation messages. The default value is 4. |
| NXDUO_DESTINATION_TABLE_SIZE | Renamed to ***NX_IPV6_DESTINATION_TABLE_SIZE***. Although it is still being supported, new designs are encouraged to use ***NX_IPV6_DESTINATION_TABLE_SIZE***. |
| NXDUO_DISABLE_ICMPV6_ERROR_MESSAGE | Renamed to ***NX_DISABLE_ICMPV6_ERROR_MESSAGE***. Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_ICMPV6_ERROR_MESSAGE***. |

| Option | Description |
| --- | --- |
| NXDUO_DISABLE_ICMPV6_REDIRECT_PROCESS | Renamed to ***NX_DISABLE_ICMPV6_REDIRECT_PROCESS***. Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_ICMPV6_REDIRECT_PROCESS*** |
| NXDUO_DISABLE_ICMPV6_ROUTER_ADVERTISEMENT_PROCESS | Renamed to ***NX_DISABLE_ICMPV6_ROUTER_ADVERTISEMENT_*** Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_ICMPV6_ROUTER_ADVERTISEMENT_*** |
| NXDUO_DISABLE_ICMPV6_ROUTER_SOLICITATION | Renamed to ***NX_DISABLE_ICMPV6_ROUTER_SOLICITATION***. Although it is still being supported, new designs are encouraged to use ***NX_DISABLE_ICMPV6_ROUTER_SOLICITATION***. |
| NXDUO_ICMPV6_MAX_RTR_SOLICITATIONS | Renamed to ***NX_ICMPV6_MAX_RTR_SOLICITATIONS***. Although it is still being supported, new designs are encouraged to use ***NX_ICMPV6_MAX_RTR_SOLICITATIONS***. |
| NXDUO_ICMPV6_RTR_SOLICITATION_INTERVAL | Renamed to ***NX_ICMPV6_RTR_SOLICITATION_INTERVAL***. This symbol is being depreciated. Although it is still being supported, new designs are encouraged to use ***NX_ICMPV6_RTR_SOLICITATION_INTERVAL*** |

## NetX Duo Version ID

The current version of NetX Duo is available to both the user and the application software during runtime. You can obtain the NetX Duo version from examination of the **nx_port.h** file. In addition, this file also contains a version history of the corresponding port. Application software can obtain the NetX Duo version by examining the global string ***_nx_version_id*** in ***nx_port.h***.

Application software can also obtain release information from the constants shown below defined in ***nx_api.h***.

These constants identify the current product release by name and the product major and minor version.

```
#define EL_PRODUCT_NETXDUO
#define NETXDUO_MAJOR_VERSION
#define NETXDUO_MINOR_VERSION
```