Chapter 4 - Description of NetX Duo Services

Contents

Chapter 4 - Description of NetX Duo Services

This chapter contains a description of all NetX Duo services in alphabetic order. Service names are designed so all similar services are grouped together. For example, all ARP services are found at the beginning of this chapter.

There are numerous new services in NetX Duo introduced to support IPv6-based protocols and operations. IPv6-enabled services in Net Duo have the prefix ***nxd***, indicating that they are designed for IPv4 and IPv6 dual stack operation.

Existing services in NetX are fully supported in NetX Duo. NetX applications can be migrated to NetX Duo with minimal porting effort.

> **Note:** *Note that a BSD-Compatible Socket API is available for legacy application code that cannot take full advantage of the high-performance NetX Duo API. Refer to Appendix D for more information on the BSD-Compatible Socket API.*

In the **Return Values** section of each description, values in **BOLD** are not affected by the **NX_DISABLE_ERROR_CHECKING** option used to disable the API error checking, while values in non-bold are completely disabled. The "Allowed From" sections indicate from which each NetX Duo service can be called.

nx_arp_dynamic_entries_invalidate

Invalidate all dynamic entries in the ARP cache

Prototype

```
UINT nx_arp_dynamic_entries_invalidate(NX_IP *ip_ptr);
```

Description

This service invalidates all dynamic ARP entries currently in the ARP cache.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP cache invalidate.
- **NX_NOT_ENABLED** (0x14) ARP is not enabled.
- **NX_PTR_ERROR** (0x07) Invalid IP address.
- **NX_CALLER_ERROR** (0x11) Caller is not a thread.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Invalidate all dynamic entries in the ARP cache. */
status = nx_arp_dynamic_entries_invalidate(&ip_0);

/* If status is NX_SUCCESS the dynamic ARP entries were
   successfully invalidated. */
```

See Also

- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_dynamic_entry_set

Set dynamic ARP entry

Prototype

```
UINT nx_arp_dynamic_entry_set(
    NX_IP *ip_ptr,
    ULONG ip_address,
    ULONG physical_msw,
    ULONG physical_lsw);
```

Description

This service allocates a dynamic entry from the ARP cache and sets up the specified IP to physical address mapping. If a zero physical address is specified, an actual ARP request is sent to the network in order to have the physical address resolved. Also note that this entry will be removed if ARP aging is active or if the ARP cache is exhausted and this is the least recently used ARP entry.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: IP address to map.
- *physical_msw*: Top 16 bits (47-32) of the physical address.
- *physical_lsw*: Lower 32 bits (31-0) of the physical address.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP dynamic entry set.
- **NX_NO_MORE_ENTRIES** (0x17) No more ARP entries are available in the ARP cache.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_PTR_ERROR** (0x07) Invalid IP instance pointer.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Setup a dynamic ARP entry on the previously created IP
   Instance 0. */
status = nx_arp_dynamic_entry_set(&ip_0, IP_ADDRESS(1,2,3,4),
                                  0x1022, 0x1234);
```

```
/* If status is NX_SUCCESS, there is now a dynamic mapping between
   the IP address of 1.2.3.4 and the physical hardware address of
   10:22:00:00:12:34. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_enable

Enable Address Resolution Protocol (ARP)

Prototype

```
UINT nx_arp_enable(
    NX_IP *ip_ptr,
    VOID *arp_cache_memory,
    ULONG arp_cache_size);
```

Description

This service initializes the ARP component of NetX Duo for the specific IP instance. ARP initialization includes setting up the ARP cache and various ARP processing routines necessary for sending and receiving ARP messages.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *arp_cache_memory*: Pointer to memory area to place ARP cache.
- *arp_cache_size*: Each ARP entry is 52 bytes, the total number of ARP entries is, therefore, the size divided by 52.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP enable.
- **NX_PTR_ERROR** (0x07) Invalid IP or cache memory pointer.
- **NX_SIZE_ERROR** (0x09) User supplied ARP cache memory is too small.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_ALREADY_ENABLED** (0x15) This component has already been enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Enable ARP and supply 1024 bytes of ARP cache memory for
   previously created IP Instance ip_0. */
status = nx_arp_enable(&ip_0, (void *) pointer, 1024);

/* If status is NX_SUCCESS, ARP was successfully enabled for this IP
instance.*/
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_entry_delete

Delete an ARP entry

Prototype

```
UINT nx_arp_entry_delete(
    NX_IP *ip_ptr,
    ULONG ip_address);
```

Description

This service removes an ARP entry for the given IP address from its IP internal ARP table.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: ARP entry with the specified IP address should be deleted.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP enable.
- **NX_ENTRY_NOT_FOUND** (0x16) No entry with the specified IP address can be found.
- **NX_PTR_ERROR** (0x07) Invalid IP or cache memory pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_IP_ADDRESS_ERROR** (0x21) Specified IP address is invalid.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Delete the ARP entry with the IP address 1.2.3.4. */
status = nx_arp_entry_delete(&ip_0, IP_ADDRESS(1, 2, 3, 4));

/* If status is NX_SUCCESS, ARP entry with the specified IP address
   is deleted.*/
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find

- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_gratuitous_send

Send gratuitous ARP request

Prototype

```
UINT nx_arp_gratuitous_send(
    NX_IP *ip_ptr,
    VOID (*response_handler)(NX_IP *ip_ptr, NX_PACKET *packet_ptr));
```

Description

This service goes through all the physical interfaces to transmit gratuitous ARP requests as long as the interface IP address is valid. If an ARP response is subsequently received, the supplied response handler is called to process the response to the gratuitous ARP.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *response_handler*: Pointer to response handling function. If NX_NULL is supplied, responses are ignored.

Return Values

- **NX_SUCCESS** (0x00) Successful gratuitous ARP send.
- **NX_NO_PACKET** (0x01) No packet available.
- **NX_NOT_ENABLED** (0x14) ARP is not enabled.
- **NX_IP_ADDRESS_ERROR** (0x21) Current IP address is invalid.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Caller is not a thread.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Send gratuitous ARP without any response handler. */
status = nx_arp_gratuitous_send(&ip_0, NX_NULL);

/* If status is NX_SUCCESS the gratuitous ARP was successfully
   sent. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_hardware_address_find

Locate physical hardware address given an IP address

Prototype

```
UINT nx_arp_hardware_address_find(
    NX_IP *ip_ptr,
    ULONG ip_address,
    ULONG *physical_msw,
    ULONG *physical_lsw);
```

Description

This service attempts to find a physical hardware address in the ARP cache that is associated with the supplied IP address.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: IP address to search for.
- *physical_msw*: Pointer to the variable for returning the top 16 bits (47-32) of the physical address.
- *physical_lsw*: Pointer to the variable for returning the lower 32 bits (31-0) of the physical address.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP hardware address find.
- **NX_ENTRY_NOT_FOUND** (0x16) Mapping was not found in the ARP cache.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_PTR_ERROR** (0x07) Invalid IP or memory pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Search for the hardware address associated with the IP address of
   1.2.3.4 in the ARP cache of the previously created IP
   Instance 0. */
status = nx_arp_hardware_address_find(&ip_0, IP_ADDRESS(1,2,3,4),
                                      &physical_msw,
                                      &physical_lsw);

/* If status is NX_SUCCESS, the variables physical_msw and
   physical_lsw contain the hardware address.*/
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_info_get
- nx_arp_ip_address_find

- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_info_get

Retrieve information about ARP activities

Prototype

```
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);
```

Description

This service retrieves information about ARP activities for the associated IP instance.

> **Note:** *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *arp_requests_sent*: Pointer to destination for the total ARP requests sent from this IP instance.
- *arp_requests_received*: Pointer to destination for the total ARP requests received from the network.
- *arp_responses_sent*: Pointer to destination for the total ARP responses sent from this IP instance.
- *arp_responses_received*: Pointer to the destination for the total ARP responses received from the network.
- *arp_dynamic_entries*: Pointer to the destination for the current number of dynamic ARP entries.

- *arp_static_entries*: Pointer to the destination for the current number of static ARP entries.
- *arp_aged_entries*: Pointer to the destination of the total number of ARP entries that have aged and became invalid.
- *arp_invalid_messages*: Pointer to the destination of the total invalid ARP messages received.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP information retrieval.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Pickup ARP information for ip_0. */
status = nx_arp_info_get(&ip_0, &arp_requests_sent,
                         &arp_requests_received,
                         &arp_responses_sent,
                         &arp_responses_received,
                         &arp_dynamic_entries,
                         &arp_static_entries,
                         &arp_aged_entries,
                         &arp_invalid_messages);

/* If status is NX_SUCCESS, the ARP information has been stored in
   the supplied variables. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_ip_address_find
- nx_arp_static_entries_delete

- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_ip_address_find

Locate IP address given a physical address

Prototype

```
UINT nx_arp_ip_address_find(
    NX_IP *ip_ptr,
    ULONG *ip_address,
    ULONG physical_msw,
    ULONG physical_lsw);
```

Description

This service attempts to find an IP address in the ARP cache that is associated with the supplied physical address.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: Pointer to return IP address, if one is found that has been mapped.
- *physical_msw*: Top 16 bits (47-32) of the physical address to search for.
- *physical_lsw*: Lower 32 bits (31-0) of the physical address to search for.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP IP address find
- **NX_ENTRY_NOT_FOUND** (0x16) Mapping was not found in the ARP cache.
- **NX_PTR_ERROR** (0x07) Invalid IP or memory pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_INVALID_PARAMETERS** (0x4D) Physical_msw and physical_lsw are both 0.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Search for the IP address associated with the hardware address of
   0x0:0x01234 in the ARP cache of the previously created IP
   Instance ip_0. */
status = nx_arp_ip_address_find(&ip_0, &ip_address, 0x0, 0x1234);

/* If status is NX_SUCCESS, the variables ip_address contains the
   associated IP address. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_static_entries_delete

Delete all static ARP entries

Prototype

```
UINT nx_arp_static_entries_delete(NX_IP *ip_ptr);
```

Description

This service deletes all static entries in the ARP cache.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Static entries are deleted.
- **NX_PTR_ERROR** (0x07) Invalid ip_ptr pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Delete all the static ARP entries for IP Instance 0, assuming
   "ip_0" is the NX_IP structure for IP Instance 0. */
status = nx_arp_static_entries_delete(&ip_0);

/* If status is NX_SUCCESS all static ARP entries in the ARP cache
   have been deleted. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_static_entry_create

Create static IP to hardware mapping in ARP cache

Prototype

```
UINT nx_arp_static_entry_create(
    NX_IP *ip_ptr,
    ULONG ip_address,
    ULONG physical_msw,
    ULONG physical_lsw);
```

Description

This service creates a static IP-to-physical address mapping in the ARP cache for the specified IP instance. Static ARP entries are not subject to ARP periodic updates.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: IP address to map.
- *physical_msw*: Top 16 bits (47-32) of the physical address to map.
- *physical_lsw*: Lower 32 bits (31-0) of the physical address to map.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP static entry create.
- **NX_NO_MORE_ENTRIES** (0x17) No more ARP entries are available in the ARP cache.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_INVALID_PARAMETERS** (0x4D) Physical_msw and physical_lsw are both 0.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Create a static ARP entry on the previously created IP
   Instance 0. */
status = nx_arp_static_entry_create(&ip_0, IP_ADDRESS(1,2,3,4),
                                    0x0, 0x1234);
```

```
/* If status is NX_SUCCESS, there is now a static mapping between
   the IP address of 1.2.3.4 and the physical hardware address of
   0x00:0x1234. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_arp_static_entry_delete

Delete static IP to hardware mapping in ARP cache

Prototype

```
UINT nx_arp_static_entry_delete(
    NX_IP *ip_ptr,
    ULONG ip_address,
    ULONG physical_msw,
    ULONG physical_lsw);
```

Description

This service finds and deletes a previously created static IP-to-physical address mapping in the ARP cache for the specified IP instance.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: IP address that was mapped statically.
- *physical_msw*: Top 16 bits (47 - **32) of the physical address that was mapped statically.
- *physical_lsw*: Lower 32 bits (31 - **0) of the physical address that was mapped statically.

Return Values

- **NX_SUCCESS** (0x00) Successful ARP static entry delete.
- **NX_ENTRY_NOT_FOUND** (0x16) Static ARP entry was not found in the ARP cache.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_INVALID_PARAMETERS** (0x4D) Physical_msw and physical_lsw are both 0.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Delete a static ARP entry on the previously created IP
   instance ip_0. */
status = nx_arp_static_entry_delete(&ip_0, IP_ADDRESS(1,2,3,4),
                                    0x0, 0x1234);

/* If status is NX_SUCCESS, the previously created static ARP entry
   was successfully deleted. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nx_icmp_enable

Enable Internet Control Message Protocol (ICMP)

Prototype

```
UINT nx_icmp_enable(NX_IP *ip_ptr);
```

Description

This service enables the ICMP component for the specified IP instance. The ICMP component is responsible for handling Internet error messages and ping requests and replies.

> **Important:** *This service only enables ICMP for IPv4 service. To enable both ICMPv4 and ICMPv6, applications shall use the* ***nxd_icmp_enable*** *service.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful ICMP enable.
- **NX_ALREADY_ENABLED** (0x15) ICMP is already enabled.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Enable ICMP on the previously created IP Instance ip_0. */
status = nx_icmp_enable(&ip_0);

/* If status is NX_SUCCESS, ICMP is enabled. */
```

See Also

- nx_icmp_info_get
- nx_icmp_ping
- nxd_icmp_enable

- nxd_icmp_ping
- nxd_icmp_source_ping
- nxd_icmpv6_ra_flag_callback_set

nx_icmp_info_get

Retrieve information about ICMP activities

Prototype

```
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);
```

Description

This service retrieves information about ICMP activities for the specified IP instance.

> [!NOTE]
> *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *pings_sent*: Pointer to destination for the total number of pings sent.
- *ping_timeouts*: Pointer to destination for the total number of ping timeouts.
- *ping_threads_suspended*: Pointer to destination of the total number of threads suspended on ping requests.
- *ping_responses_received*: Pointer to destination of the total number of ping responses received.
- *icmp_checksum_errors*: Pointer to destination of the total number of ICMP checksum errors.
- *icmp_unhandled_messages*: Pointer to destination of the total number of un-handled ICMP messages.

Return Values

- **NX_SUCCESS** (0x00) Successful ICMP information retrieval.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.

- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Retrieve ICMP information from previously created IP
   instance ip_0. */
status = nx_icmp_info_get(&ip_0, &pings_sent, &ping_timeouts,
                          &ping_threads_suspended,
                          &ping_responses_received,
                          &icmp_checksum_errors,
                          &icmp_unhandled_messages);
/* If status is NX_SUCCESS, ICMP information was retrieved. */
```

See Also

- nx_icmp_enable
- nx_icmp_ping
- nxd_icmp_enable
- nxd_icmp_ping
- nxd_icmp_source_ping
- nxd_icmpv6_ra_flag_callback_set

nx_icmp_ping

Send ping request to specified IP address

Prototype

```
UINT nx_icmp_ping(
    NX_IP *ip_ptr,
    ULONG ip_address,
    CHAR *data, ULONG data_size,
    NX_PACKET **response_ptr,
    ULONG wait_option);
```

Description

This service sends a ping request to the specified IP address and waits for the specified amount of time for a ping response message. If no response is received,

an error is returned. Otherwise, the entire response message is returned in the variable pointed to by response_ptr.

To send a ping request to an IPv6 destination, applications shall use the ***nxd_icmp_ping*** or ***nxd_icmp_source_ping*** service.

> **Warning:** *If NX_SUCCESS is returned, the application is responsible for releasing the received packet after it is no longer needed.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: IP address, in host byte order, to ping.
- *data*: Pointer to data area for ping message.
- *data_size*: Number of bytes in the ping data
- *response_ptr*: Pointer to packet pointer to return the ping response message in.
- *wait_option*: Defines the number of ThreadX timer ticks to wait for a ping response. The wait options are defined as follows:

Wait Option	Value
NX_NO_WAIT	(0x00000000)
timeout value in ticks	(0x00000001 through 0xFFFFFFFE)
NX_WAIT_FOREVER	0xFFFFFFFF

Return Values

- **NX_SUCCESS** (0x00) Successful ping. Response message pointer was placed in the variable pointed to by response_ptr.
- **NX_NO_PACKET** (0x01) Unable to allocate a ping request packet.
- **NX_OVERFLOW** (0x03) Specified data area exceeds the default packet size for this IP instance.
- **NX_NO_RESPONSE** (0x29) Requested IP did not respond.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_PTR_ERROR** (0x07) Invalid IP or response pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Issue a ping to IP address 1.2.3.5 from the previously created IP
   Instance ip_0. */
status = nx_icmp_ping(&ip_0, IP_ADDRESS(1,2,3,5), "abcd", 4,
                      &response_ptr, 10);

/* If status is NX_SUCCESS, a ping response was received from IP
   address 1.2.3.5 and the response packet is contained in the
   packet pointed to by response_ptr. It should have the same "abcd"
   four bytes of data. */
```

See Also

- nx_icmp_enable
- nx_icmp_info_get
- nxd_icmp_enable
- nxd_icmp_ping
- nxd_icmp_source_ping
- nxd_icmpv6_ra_flag_callback_set

nx_igmp_enable

Enable Internet Group Management Protocol (IGMP)

Prototype

```
UINT nx_igmp_enable(NX_IP *ip_ptr);
```

Description

This service enables the IGMP component on the specified IP instance. The IGMP component is responsible for providing support for IP multicast group management operations.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IGMP enable.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_ALREADY_ENABLED** (0x15) This component has already been enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Enable IGMP on the previously created IP Instance ip_0. */
status = nx_igmp_enable(&ip_0);

/* If status is NX_SUCCESS, IGMP is enabled. */
```

See Also

- nx_igmp_info_get
- nx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_igmp_info_get

Retrieve information about IGMP activities

Prototype

```
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);
```

Description

This service retrieves information about IGMP activities for the specified IP instance.

Important: *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *igmp_reports_sent*: Pointer to destination for the total number of ICMP reports sent.
- *igmp_queries_received*: Pointer to destination for the total number of queries received by multicast router.
- *igmp_checksum_errors*: Pointer to destination of the total number of IGMP checksum errors on receive packets.
- *current_groups_joined*: Pointer to destination of the current number of groups joined through this IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IGMP information retrieval.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Retrieve IGMP information from previously created IP Instance ip_0. */
status = nx_igmp_info_get(&ip_0, &igmp_reports_sent,
                          &igmp_queries_received,
                          &igmp_checksum_errors,
                          &current_groups_joined);

/* If status is NX_SUCCESS, IGMP information was retrieved. */
```

See Also

- nx_igmp_enable
- nx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join

- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_igmp_loopback_disable

Disable IGMP loopback

Prototype

```
UINT nx_igmp_loopback_disable(NX_IP *ip_ptr);
```

Description

This service disables IGMP loopback for all subsequent multicast groups joined.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IGMP loopback disable.
- **NX_NOT_ENABLED** (0x14) IGMP is not enabled.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Caller is not a thread or initialization.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Disable IGMP loopback for all subsequent multicast groups
   joined. */
status = nx_igmp_loopback_disable(&ip_0);

/* If status is NX_SUCCESS IGMP loopback is disabled. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_get
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_igmp_loopback_enable

Enable IGMP loopback

Prototype

```
UINT nx_igmp_loopback_enable(NX_IP *ip_ptr);
```

Description

This service enables IGMP loopback for all subsequent multicast groups joined.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IGMP loopback disable.
- **NX_NOT_ENABLED** (0x14) IGMP is not enabled.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Caller is not a thread or initialization.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Enable IGMP loopback for all subsequent multicast
   groups joined. */
status = nx_igmp_loopback_enable(&ip_0);

/* If status is NX_SUCCESS IGMP loopback is enabled. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_getnx_igmp_loopback_disable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_igmp_multicast_interface_join

Join IP instance to specified multicast group via an interface

Prototype

```
UINT nx_igmp_multicast_interface_join(
    NX_IP *ip_ptr,
    ULONG group_address,
    UINT interface_index);
```

Description

This service joins an IP instance to the specified multicast group via a specified network interface. An internal counter is maintained to keep track of the number of times the same group has been joined. After joining the multicast group, the IGMP component will allow reception of IP packets with this group address via the specified network interface and also report to routers that this IP is a member of this multicast group. The IGMP membership join, report, and leave messages are also sent via the specified network interface. To join an IPv4 multicast group without sending IGMP group membership report, application shall use the service ***nx_ipv4_multicast_interface_join***.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

- *group_address*: Class D IP multicast group address to join in host byte order.
- *interface_index*: Index of the Interface attached to the NetX Duo instance.

Return Values

- **NX_SUCCESS** (0x00) Successful multicast group join.
- **NX_NO_MORE_ENTRIES** (0x17) No more multicast groups can be joined, maximum exceeded.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.
- **NX_IP_ADDRESS_ERROR** (0x21) Multicast group address provided is not a valid class D address.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) IP multicast support is not enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Previously created IP Instance joins the multicast group
   244.0.0.200, via the interface at index 1 in the IP interface
   list. */
#define INTERFACE_INDEX 1
status = nx_igmp_multicast_interface_join
                                  (&ip IP_ADDRESS(244,0,0,200),
                                   INTERFACE_INDEX);

/* If status is NX_SUCCESS, the IP instance has successfully joined
   the multicast group. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_getnx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join

- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_igmp_multicast_interface_leave

Leave specified multicast group via an interface

Prototype

```
UINT nx_igmp_multicast_interface_leave(
    NX_IP *ip_ptr,
    ULONG group_address,
    UINT interface_index);
```

Description

This service leaves the specified multicast group via a specified network interface. An internal counter is maintained to keep track of the number of times the same group has been a member of. After leaving the multicast group, the IGMP component will send out proper membership report, and may leave the group if there are no members from this node. To leave an IPv4 multicast group without sending IGMP group membership report, application shall use the service ***nx_ipv4_multicast_interface_leave***.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *group_address*: Class D IP multicast group address to leave. The IP address is in host byte order.
- *interface_index*: Index of the Interface attached to the NetX Duo instance.

Return Values

- **NX_SUCCESS** (0x00) Successful multicast group join.
- **NX_ENTRY_NOT_FOUND** (0x16) The specified multicast group address cannot be found in the local multicast table.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.
- **NX_IP_ADDRESS_ERROR** (0x21) Multicast group address provided is not a valid class D address.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) IP multicast support is not enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Leave the multicast group 244.0.0.200. */
#define INTERFACE_INDEX 1
status = nx_igmp_multicast_interface_leave
                                (&ip IP_ADDRESS(244,0,0,200),
                                 INTERFACE_INDEX);

/* If status is NX_SUCCESS, the IP instance has successfully leaves
   the multicast group 244.0.0.200. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_getnx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_igmp_multicast_join

Join IP instance to specified multicast group

Prototype

```
UINT nx_igmp_multicast_join(
    NX_IP *ip_ptr,
    ULONG group_address);
```

Description

This service joins an IP instance to the specified multicast group. An internal counter is maintained to keep track of the number of times the same group has been joined. The driver is commanded to send an IGMP report if this is the first join request out on the network indicating the host's intention to join the group. After joining, the IGMP component will allow reception of IP packets with this group address and report to routers that this IP is a member of this multicast group. To join an IPv4 multicast group without sending IGMP group membership report, application shall use the service ***nx_ipv4_multicast_interface_join***.

[!NOTE]
To join a multicast group on a non-primary device, use the service ***nx_igmp_multicast_interface_join.***

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *group_address*: Class D IP multicast group address to join.

Return Values

- **NX_SUCCESS** (0x00) Successful multicast group join.
- **NX_NO_MORE_ENTRIES** (0x17) No more multicast groups can be joined, maximum exceeded.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP group address.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Previously created IP Instance ip_0 joins the multicast group
   224.0.0.200. */
status = nx_igmp_multicast_join(&ip_0, IP_ADDRESS(224,0,0,200);

/* If status is NX_SUCCESS, this IP instance has successfully
   joined the multicast group 224.0.0.200. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_get
- nx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave

- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_igmp_multicast_leave

Cause IP instance to leave specified multicast group

Prototype

```
UINT nx_igmp_multicast_leave(
    NX_IP *ip_ptr,
    ULONG group_address);
```

Description

This service causes an IP instance to leave the specified multicast group, if the number of leave requests matches the number of join requests. Otherwise, the internal join count is simply decremented. To leave an IPv4 multicast group without sending IGMP group membership report, application shall use the service ***nx_ipv4_multicast_interface_leave***.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *group_address*: Multicast group to leave.

Return Values

- **NX_SUCCESS** (0x00) Successful multicast group join.
- **NX_ENTRY_NOT_FOUND** (0x16) Previous join request was not found.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP group address.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Cause IP instance to leave the multicast group 224.0.0.200. */
status = nx_igmp_multicast_leave(&ip_0, IP_ADDRESS(224,0,0,200);

/* If status is NX_SUCCESS, this IP instance has successfully left
   the multicast group 224.0.0.200. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_get
- nx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_ip_address_change_notifiy

Notify application if IP address changes

Prototype

```
UINT nx_ip_address_change_notify(
    NX_IP *ip_ptr,
    VOID(*change_notify)(NX_IP *, VOID *),
    VOID *additional_info);
```

Description

This service registers an application notification function that is called whenever the IPv4 address is changed.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *change_notify*: Pointer to IP change notification function. If this parameter is NX_NULL, IP address change notification is disabled.
- *additional_info*: Pointer to optional additional information that is also supplied to the notification function when the IP address is changed.

Return Values

- **NX_SUCCESS** (0x00) Successful IP address change notification.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Register the function "my_ip_changed" to be called whenever the
   IP address is changed. */
status = nx_ip_address_change_notify(&ip_0, my_ip_changed,
                                     NX_NULL);

/* If status is NX_SUCCESS, the "my_ip_changed" function will be
   called whenever the IP address changes. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable

- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_address_get

Retrieve IPv4 address and network mask

Prototype

```
UINT nx_ip_address_get(
    NX_IP *ip_ptr,
    ULONG *ip_address,
    ULONG *network_mask);
```

Description

This service retrieves IPv4 address and its subnet mask of the primary network interface.

> **Important:** *To obtain information of the secondary device, use the service* ***nx_ip_interface_address_get****.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: Pointer to destination for IP address.
- *network_mask*: Pointer to destination for network mask.

Return Values

- **NX_SUCCESS** (0x00) Successful IP address get.
- **NX_PTR_ERROR** (0x07) Invalid IP or return variable pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Get the IP address and network mask from the previously created
   IP Instance ip_0. */
status = nx_ip_address_get(&ip_0, &ip_address, &network_mask);
```

```
/* If status is NX_SUCCESS, the variables ip_address and
   network_mask contain the IP and network mask respectively. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_address_set

Set IPv4 address and network mask

Prototype

```
UINT nx_ip_address_set(
    NX_IP *ip_ptr,
    ULONG ip_address,
    ULONG network_mask);
```

Description

This service sets IPv4 address and network mask for the primary network interface.

> **Important:** *To set IP address and network mask for the secondary device, use the service* ***nx_ip_interface_address_set****.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: New IP address.
- *network_mask*: New network mask.

Return Values

- **NX_SUCCESS** (0x00) Successful IP address set.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Set the IP address and network mask to 1.2.3.4 and 0xFFFFFF00 for
   the previously created IP Instance ip_0. */
status = nx_ip_address_set(&ip_0, IP_ADDRESS(1,2,3,4),
                           0xFFFFFF00UL);

/* If status is NX_SUCCESS, the IP instance now has an IP address of
   1.2.3.4 and a network mask of 0xFFFFFF00. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize

- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_auxiliary_packet_pool_set

Configure an auxiliary packet pool

Prototype

```
UINT nx_ip_auxiliary_packet_pool_set(
    NX_IP *ip_ptr,
    NX_PACKET_POOL *aux_pool);
```

Description

This service configures an auxiliary packet pool in the IP instance. For a memory-constrained system, the user may increase memory efficiency by creating the default packet pool with packet size of MTU, and creating an auxiliary packet pool with smaller packet size for the IP thread to transmit small packets with. The recommended packet size for the auxiliary pool is 256 bytes, assuming IPv6 and IPsec are both enabled.

By default the IP instance does not accept the auxiliary packet pool. To enable this feature, *NX_DUAL_PACKET_POOL_ENABLE* must be defined when compiling the NetX Duo library.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *aux_pool*: The auxiliary packet pool to be configured for the IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IP address set.
- **NX_NOT_SUPPORTED** (0x4B) The dual packet pool feature is not compiled in the library.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer or pool pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define SMALL_PAYLOAD_SIZE 256
NX_PACKET small_pool;

nx_packet_pool_create(&small_pool, "small pool", SMALL_PAYLOAD_SIZE,
                      small_pool_memory_ptr, small_pool_size);

/* Add the small packet pool to the IP instance. */
status = nx_ip_auxiliary_packet_pool_set(&ip_0, &small_pool);

/* If status is NX_SUCCESS, the IP instance now is able to use the
   small pool for transmitting small datagram. */
```

See Also

- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_ip_create

Create an IP instance

Prototype

```
UINT nx_ip_create(
    NX_IP *ip_ptr,
    CHAR *name, ULONG ip_address,
    ULONG network_mask,
    NX_PACKET_POOL *default_pool,
    VOID (*ip_network_driver)(NX_IP_DRIVER *),
    VOID *memory_ptr,
```

```
ULONG memory_size,
UINT priority);
```

Description

This service creates an IP instance with the user supplied IP address and network driver. In addition, the application must supply a previously created packet pool for the IP instance to use for internal packet allocation. Note that the supplied application network driver is not called until this IP's thread executes.

Parameters

- *ip_ptr*: Pointer to control block to create a new IP instance.
- *name*: Name of this new IP instance.
- *ip_address*: IP address for this new IP instance.
- *network_mask*: Mask to delineate the network portion of the IP address for sub-netting and super-netting uses.
- *default_pool*: Pointer to control block of previously created NetX Duo packet pool.
- *ip_network_driver*: User-supplied network driver used to send and receive IP packets.
- *memory_ptr*: Pointer to memory area for the IP helper thread's stack area.
- *memory_size*: Number of bytes in the memory area for the IP helper thread's stack.
- *priority*: Priority of IP helper thread.

Return Values

- **NX_SUCCESS** (0x00) Successful IP instance creation.
- **NX_NOT_IMPLEMENTED** (0x4A) NetX Duo library is configured incorrectly.
- **NX_PTR_ERROR** (0x07) Invalid IP, network driver function pointer, packet pool, or memory pointer.
- **NX_SIZE_ERROR** (0x09) The supplied stack size is too small.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_IP_ADDRESS_ERROR** (0x21) The supplied IP address is invalid.
- **NX_OPTION_ERROR** (0x21) The supplied IP thread priority is invalid.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Create an IP instance with an IP address of 1.2.3.4 and a network
   mask of 0xFFFFFF00UL. The "ethernet_driver" specifies the entry
   point of the application specific network driver and the
   "stack_memory_ptr" specifies the start of a 1024 byte memory
   area that is used for this IP instance's helper thread. */
status = nx_ip_create(&ip_0, "NetX IP Instance ip_0",
                      IP_ADDRESS(1, 2, 3, 4),
                      0xFFFFFF00UL, &pool_0, ethernet_driver,
                      stack_memory_ptr, 1024, 1);

/* If status is NX_SUCCESS, the IP instance has been created. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_delete

Delete previously created IP instance

Prototype

```
UINT nx_ip_delete(NX_IP *ip_ptr);
```

Description

This service deletes a previously created IP instance and releases all of the system resources owned by the IP instance.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IP deletion.
- **NX_SOCKETS_BOUND** (0x28) This IP instance still has UDP or TCP sockets bound to it. All sockets must be unbound and deleted prior to deleting the IP instance.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

Yes

Example

```
/* Delete a previously created IP instance. */
status = nx_ip_delete(&ip_0);

/* If status is NX_SUCCESS, the IP instance has been deleted. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable

- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_driver_direct_command

Issue command to network driver

Prototype

```
UINT nx_ip_driver_direct_command
    (NX_IP *ip_ptr,
    UINT command,
    ULONG *return_value_ptr);
```

Description

This service provides a direct interface to the application's primary network interface driver specified during the ***nx_ip_create*** call. Application-specific commands can be used providing their numeric value is greater than or equal to **NX_LINK_USER_COMMAND**.

> **Important:** *To issue command for the secondary device, use the* ***nx_ip_driver_interface_direct_command*** *service.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *command*: Numeric command code. Standard commands are defined as follows:
  - **NX_LINK_GET_STATUS** (10)
  - **NX_LINK_GET_SPEED** (11)
  - **NX_LINK_GET_DUPLEX_TYPE** (12)
  - **NX_LINK_GET_ERROR_COUNT** (13)
  - **NX_LINK_GET_RX_COUNT** (14)
  - **NX_LINK_GET_TX_COUNT** (15)

  - **NX_LINK_GET_ALLOC_ERRORS** (16)
  - **NX_LINK_USER_COMMAND** (50)
- *return_value_ptr*: Pointer to return variable in the caller.

Return Values

- **NX_SUCCESS** (0x00) Successful network driver direct command.
- **NX_UNHANDLED_COMMAND** (0x44) Unhandled or unimplemented network driver command.
- **NX_PTR_ERROR** (0x07) Invalid IP or return value pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Make a direct call to the application-specific network driver
   for the previously created IP instance. For this example, the
   network driver is interrogated for the link status. */
status = nx_ip_driver_direct_command(&ip_0, NX_LINK_GET_STATUS,
                                     &link_status);

/* If status is NX_SUCCESS, the link_status variable contains a
   NX_TRUE or NX_FALSE value representing the status of the
   physical link. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get

- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_driver_interface_direct_command

Issue command to network driver

Prototype

```
UINT nx_ip_driver_interface_direct_command(
    NX_IP *ip_ptr,
    UINT command,
    UINT interface_index,
    ULONG *return_value_ptr);
```

Description

This service provides a direct command to the application's network device driver in the IP instance. Application-specific commands can be used providing their numeric value is greater than or equal to *NX_LINK_USER_COMMAND*.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *command*: Numeric command code. Standard commands are defined as follows:
  - **NX_LINK_GET_STATUS** (10)
  - **NX_LINK_GET_SPEED** (11)
  - **NX_LINK_GET_DUPLEX_TYPE** (12)
  - **NX_LINK_GET_ERROR_COUNT** (13)
  - **NX_LINK_GET_RX_COUNT** (14)
  - **NX_LINK_GET_TX_COUNT** (15)
  - **NX_LINK_GET_ALLOC_ERRORS** (16)
  - **NX_LINK_USER_COMMAND** (50)
- *interface_index*: Index of the network interface the command should be sent to.
- *return_value_ptr*: Pointer to return variable in the caller.

Return Values

- **NX_SUCCESS** (0x00) Successful network driver direct command.
- **NX_UNHANDLED_COMMAND** (0x44) Unhandled or unimplemented network driver command.
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index
- **NX_PTR_ERROR** (0x07) Invalid IP or return value pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Make a direct call to the application-specific network driver
   for the previously created IP instance. For this example, the
   network driver is interrogated for the link status. */

/* Set the interface index to the primary device. */
UINT interface_index = 0;

status = nx_ip_driver_interface_direct_command(&ip_0,
                                               NX_LINK_GET_STATUS,
                                               interface_index,
                                               &link_status);

/* If status is NX_SUCCESS, the link_status variable contains a
   NX_TRUE or NX_FALSE value representing the status of the
   physical link. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable

- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_forwarding_disable

Disable IP packet forwarding

Prototype

```
UINT nx_ip_forwarding_disable(NX_IP *ip_ptr);
```

Description

This service disables forwarding IP packets inside the NetX Duo IP component. On creation of the IP task, this service is automatically disabled.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IP forwarding disable.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads, timers

Preemption Possible

No

Example

```
/* Disable IP forwarding on this IP instance. */
status = nx_ip_forwarding_disable(&ip_0);

/* If status is NX_SUCCESS, IP forwarding has been disabled on the
   previously created IP instance. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_forwarding_enable

Enable IP packet forwarding

Prototype

```
UINT nx_ip_forwarding_enable(NX_IP *ip_ptr);
```

Description

This service enables forwarding IP packets inside the NetX Duo IP component. On creation of the IP task, this service is automatically disabled.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IP forwarding enable.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads, timers

Preemption Possible

No

Example

```
/* Enable IP forwarding on this IP instance. */
status = nx_ip_forwarding_enable(&ip_0);

/* If status is NX_SUCCESS, IP forwarding has been enabled on the
   previously created IP instance. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable

- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_fragment_disable

Disable IP packet fragmenting

Prototype

```
UINT nx_ip_fragment_disable(NX_IP *ip_ptr);
```

Description

This service disables IPv4 and IPv6 packet fragmenting and reassembling functionality. For packets waiting to be reassembled, this service releases these packets. On creation of the IP task, this service is automatically disabled.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IP fragment disable.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) IP Fragmentation is not enabled on the IP instance.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Disable IP fragmenting on this IP instance. */
status = nx_ip_fragment_disable(&ip_0);

/* If status is NX_SUCCESS, disables IP fragmenting on the
   previously created IP instance. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_fragment_enable

Enable IP packet fragmenting

Prototype

```
UINT nx_ip_fragment_enable(NX_IP *ip_ptr);
```

Description

This service enables IPv4 and IPv6 packet fragmenting and reassembling functionality. On creation of the IP task, this service is automatically disabled.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IP fragment enable.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.

- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) IP Fragmentation features is not compiled into NetX Duo.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Enable IP fragmenting on this IP instance. */
status = nx_ip_fragment_enable(&ip_0);

/* If status is NX_SUCCESS, IP fragmenting has been enabled on the
   previously created IP instance. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_gateway_address_clear

Clear the IPv4 gateway address

Prototype

```
UINT nx_ip_gateway_address_clear(NX_IP *ip_ptr);
```

Description

This service clears the IPv4 gateway address configured in the instance. To clear an IPv6 default outer from the IP instance, applications shall use the service ***nxd_ipv6_default_router_delete.***

Parameters

- *ip_ptr*: IP control block pointer

Return Values

- **NX_SUCCESS** (0x00) Successfully cleared the IP gateway address.
- **NX_PTR_ERROR** (0x07) Invalid IP control block
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Clear the gateway address of IP instance. */
status = nx_ip_gateway_address_clear(&ip_0);

/* If status == NX_SUCCESS, the gateway address was successfully
   cleared from the IP instance. */
```

See Also

-nx_ip_gateway_address_get -nx_ip_gateway_address_set -nx_ip_info_get -nx_ip_static_route_add -nx_ip_static_route_delete -nxd_ipv6_default_router_add -nxd_ipv6_default_router_delete -nxd_ipv6_default_router_entry_get -nxd_ipv6_default_router_get -nxd_ipv6_default_router_number_of_entries_get

nx_ip_gateway_address_get

Get the IPv4 gateway address

Prototype

```
UINT nx_ip_gateway_address_get(
    NX_IP *ip_ptr,
    ULONG *ip_address);
```

Description

This service retrieves the IPv4 gateway address configured in the IP instance.

Parameters

- *ip_ptr*: IP control block pointer
- *ip_address*: Pointer to the memory where the gateway address is stored

Return Values

- **NX_SUCCESS** (0x00) Successful get
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer or ip address pointer
- **NX_NOT_FOUND** (0x4E) Gateway address not found
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
ULONG ip_address;

/* Get the gateway address of IP instance. */
status = nx_ip_gateway_address_get(&ip_0, &ip_address);

/* If status == NX_SUCCESS, the gateway address was successfully
   got. */
```

See Also

- nx_ip_gateway_address_clear
- nx_ip_gateway_address_set
- nx_ip_info_get
- nx_ip_static_route_add
- nx_ip_static_route_delete
- nxd_ipv6_default_router_add
- nxd_ipv6_default_router_delete
- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_get
- nxd_ipv6_default_router_number_of_entries_get

nx_ip_gateway_address_set

Set Gateway IP address

Prototype

```
UINT nx_ip_gateway_address_set(
    NX_IP *ip_ptr,
    ULONG ip_address);
```

Description

This service sets the IPv4 gateway IP address. All out-of-network traffic are routed to this gateway for transmission. The gateway must be directly accessible through one of the network interfaces. To configure IPv6 gateway address, use the service ***nxd_ipv6_default_router_add.***

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_address*: IP address of the gateway.

Return Values

- **NX_SUCCESS** (0x00) Successful Gateway IP address set.
- **NX_PTR_ERROR** (0x07) Invalid IP instance pointer.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, thread

Preemption Possible

No

Example

```
/* Setup the Gateway address for previously created IP
   Instance ip_0. */
status = nx_ip_gateway_address_set(&ip_0, IP_ADDRESS(1,2,3,99);

/* If status is NX_SUCCESS, all out-of-network send requests are
   routed to 1.2.3.99. */
```

See Also

- nx_ip_gateway_address_clear
- nx_ip_gateway_address_get
- nx_ip_info_get
- nx_ip_static_route_add
- nx_ip_static_route_delete
- nxd_ipv6_default_router_add
- nxd_ipv6_default_router_delete
- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_get
- nxd_ipv6_default_router_number_of_entries_get

nx_ip_info_get

Retrieve information about IP activities

Prototype

```
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);
```

Description

This service retrieves information about IP activities for the specified IP instance.

[!NOTE]
If a destination pointer is NX_NULL, that particular information is not returned to the caller.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *ip_total_packets_sent*: Pointer to destination for the total number of IP packets sent.
- *ip_total_bytes_sent*: Pointer to destination for the total number of bytes sent.
- *ip_total_packets_received*: Pointer to destination of the total number of IP receive packets.
- *ip_total_bytes_received*: Pointer to destination of the total number of IP bytes received.
- *ip_invalid_packets*: Pointer to destination of the total number of invalid IP packets.
- *ip_receive_packets_dropped*: Pointer to destination of the total number of receive packets dropped.
- *ip_receive_checksum_errors*: Pointer to destination of the total number of checksum errors in receive packets.
- *ip_send_packets_dropped*: Pointer to destination of the total number of send packets dropped.
- *ip_total_fragments_sent*: Pointer to destination of the total number of fragments sent.
- *ip_total_fragments_received*: Pointer to destination of the total number of fragments received.

Return Values

- **NX_SUCCESS** (0x00) Successful IP information retrieval.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Retrieve IP information from previously created IP
   Instance 0. */
status = nx_ip_info_get(&ip_0,
```

```
                                &ip_total_packets_sent,
                                &ip_total_bytes_sent,
                                &ip_total_packets_received,
                                &ip_total_bytes_received,
                                &ip_invalid_packets,
                                &ip_receive_packets_dropped,
                                &ip_receive_checksum_errors,
                                &ip_send_packets_dropped,
                                &ip_total_fragments_sent,
                                &ip_total_fragments_received);

/* If status is NX_SUCCESS, IP information was retrieved. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_interface_address_get

Retrieve interface IP address

Prototype

```
UINT nx_ip_interface_address_get (
    NX_IP *ip_ptr,
    UINT interface_index,
    ULONG *ip_address,
    ULONG *network_mask);
```

Description

This service retrieves the IPv4 address of a specified network interface. To retrieve IPv6 address, application shall use the service ***nxd_ipv6_address_get***

> **Caution:** *The specified device, if not the primary device, must be previously attached to the IP instance.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Interface index, the same value as the index to the network interface attached to the IP instance.
- *ip_address*: Pointer to destination for the device interface IP address.
- *network_mask*: Pointer to destination for the device interface network mask.

Return Values

- **NX_SUCCESS** (0x00) Successful IP address get.
- **NX_INVALID_INTERFACE** (0x4C) Specified network interface is invalid.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define INTERFACE_INDEX 1

/* Get device IP address and network mask for the specified
   interface index 1 in IP instance list of interfaces). */
status = nx_ip_interface_address_get(ip_ptr,INTERFACE_INDEX,
```

```
                                        &ip_address,
                                        &network_mask);

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

See Also

- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_address_mapping_configure

Configure whether address mapping is needed

Prototype

```
UINT nx_ip_interface_address_mapping_configure(
    NX_IP *ip_ptr,
    UINT interface_index,
    UINT mapping_needed);
```

Description

This service configures whether IP address to MAC address mapping is needed for the specified network interface. This service is typically called from the interface device driver to notify the IP stack whether the underlying interface requires IP address to layer two (MAC) address mapping.

Parameters

- *ip_ptr*: IP control block pointer
- *interface_index*: Index to the network interface
- *mapping_needed*: NX_TRUE – address mapping needed NX_FALSE – address mapping not needed

Return Values

- **NX_SUCCESS** (0x00) Successful configure
- **NX_INVALID_INTERFACE** (0x4C) Device index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Thread

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0
UCHAR mapping_needed = NX_TRUE;

/* Configure address mapping needed specified interface. */
status = nx_ip_interface_address_mapping_configure(&ip_0,
                                                   PRIMARY_INTERFACE,
                                                   mapping_needed);

/* If status == NX_SUCCESS, the address mapping needed was
   successfully configured. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_address_set

Set interface IP address and network mask

Prototype

```
UINT nx_ip_interface_address_set(
    NX_IP *ip_ptr,
    UINT interface_index,
    ULONG ip_address,
    ULONG network_mask);
```

Description

This service sets the IPv4 address and network mask for the specified IP interface. To configure IPv6 interface address, application shall use the service ***nxd_ipv6_address_set***.

> **Warning:** *The specified interface must be previously attached to the IP instance.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the interface attached to the NetX Duo instance.
- *ip_address*: New network interface IP address.
- *network_mask*: New interface network mask.

Return Values

- **NX_SUCCESS** (0x00) Successful IP address set.
- **NX_INVALID_INTERFACE** (0x4C) Specified network interface is invalid.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid pointers.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define INTERFACE_INDEX 1

/* Set device IP address and network mask for the specified
   interface index 1 in IP instance list of interfaces). */
status = nx_ip_interface_address_set(ip_ptr, INTERFACE_INDEX,
```

```
                                        ip_address,
                                        network_mask);

/* If status is NX_SUCCESS the interface IP address and mask was
   successfully set. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_attach

Attach network interface to IP instance

Prototype

```
UINT nx_ip_interface_attach(
    NX_IP *ip_ptr,
    CHAR *interface_name,
    ULONG ip_address,
    ULONG network_mask,
    VOID(*ip_link_driver)(struct NX_IP_DRIVER_STRUCT *));
```

Description

This service adds a physical network interface to the IP interface. Note the IP instance is created with the primary interface so each additional interface is secondary to the primary interface. The total number of network interfaces attached to the IP instance (including the primary interface) cannot exceed **NX_MAX_PHYSICAL_INTERFACES**.

If the IP thread has not been running yet, the secondary interfaces will be initialized as part of the IP thread startup process that initializes all physical interfaces.

If the IP thread is not running yet, the secondary interface is initialized as part of the ***nx_ip_interface_attach*** service.

Warning: *ip_ptr must point to a valid NetX Duo IP structure.* ***NX_MAX_PHYSICAL_INTERFACES*** *must be configured for the number of network interfaces for the IP instance. The default value is one.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_name*: Pointer to interface name string.
- *ip_address*: Device IP address in host byte order.
- *network_mask*: Device network mask in host byte order.
- *ip_link_driver*: Ethernet driver for the interface.

Return Values

- **NX_SUCCESS** (0x00) Entry is added to static routing table.
- **NX_NO_MORE_ENTRIES** (0x17) Max number of interfaces. NX_MAX_PHYSICAL_INTERFACES is exceeded. If IPv6 is enabled, this error may also indicate that the driver may not have enough resource to handle IPv6 multicast operations.
- **NX_DUPLICATED_ENTRY** (0x52) The supplied IP address is already used on this IP instance.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid pointer input.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address input.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Attach secondary device for device IP address 192.168.1.68 with
   the specified Ethernet driver. */
status = nx_ip_interface_attach(ip_ptr, "secondary_port",
                                IP_ADDRESS(192,168,1,68),
                                0xFFFFFF00UL,
                                nx_etherDriver);

/* If status is NX_SUCCESS the interface was successfully added to
   the IP instance interface table. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_capability_get

Get interface hardware capability

Prototype

```
UINT nx_ip_interface_capability_get(
    NX_IP *ip_ptr,
    UINT interface_index,
    ULONG *interface_capability_flag);
```

Description

This service retrieves the capability flag from the specified network interface. To use this service, the NetX Duo library must be built with the option ***NX_ENABLE_INTERFACE_CAPABILITY*** enabled.

Parameters

- *ip_ptr*: IP control block pointer
- *interface_index*: Index of the network interface
- *interface_capability_flag*: Pointer to memory space for the capability flag

Return Values

- **NX_SUCCESS** (0x00) Successfully obtained interface capability information.
- **NX_NOT_SUPPORTED** (0x4B) Interface capability feature is not supported in this build.
- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer or Invalid capability flag pointer

- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0
ULONG       capability_flag;

/* Get the hardware capability flag of specified interface. */
status = nx_ip_interface_capability_get(&ip_0,
                                        PRIMARY_INTERFACE,
                                        &capability_flag);

/* If status == NX_SUCCESS, the capability flag from the primary
   interface was successfully retrieved. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_capability_set

Set the hardware capability flag

Prototype

```
UINT nx_ip_interface_capability_set(
    NX_IP *ip_ptr,
```

```
    UINT interface_index,
    ULONG interface_capability_flag);
```

Description

This service is used by the network device driver to configure the capability flag for a specified network interface. To use this service, the NetX Duo library must be compiled with the option ***NX_ENABLE_INTERFACE_CAPABILITY*** defined.

Parameters

- *ip_ptr*: IP control block pointer
- *interface_index*: Index of network interface
- *interface_capability_flag*: Capability flag for output

Return Values

- **NX_SUCCESS** (0x00) Successfully set interface hardware capability flag.
- **NX_NOT_SUPPORTED** (0x4B) Interface capability feature is not supported in this build.
- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0
ULONG capability_flag = \
                NX_INTERFACE_CAPABILITY_IPV4_TX_CHECKSUM |\
                NX_INTERFACE_CAPABILITY_IPV4_RX_CHECKSUM;
UINT device_index = 0;

/* Set the hardware capability flag of specified interface. */
status = nx_ip_interface_capability_set(&ip_0,
                                        PRIMARY_INTERFACE,
                                        capability_flag);
```

```
/* If status == NX_SUCCESS, the hardware capability flag was
   successfully set. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_detach

Detach the specified interface from the IP instance

Prototype

```
UINT nx_ip_interface_address_set(
    NX_IP *ip_ptr,
    UINT index);
```

Description

This service detaches the specified IP interface from the IP instance. Once an interface is detached, all connected TCP sockets closed, and ND cache and ARP entries for this interface are removed from their respective tables. IGMP memberships for this interface are removed.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *index*: Index of the interface to be removed.

Return Values

- **NX_SUCCESS** (0x00) Successfully removed a physical interface.
- **NX_INVALID_INTERFACE** (0x4C) Specified network interface is invalid.
- **NX_PTR_ERROR** (0x07) Invalid pointers.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define INTERFACE_INDEX 1

/* Detach interface 1. */
status = nx_ip_interface_detach(&IP_0, INTERFACE_INDEX);

/* If status is NX_SUCCESS the interface is successfully detached
   from the IP instance. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_info_get

Retrieve network interface parameters

Prototype

```
UINT nx_ip_interface_info_get(
    NX_IP *ip_ptr,
    UINT interface_index,
    CHAR **interface_name,
    ULONG *ip_address,
    ULONG *network_mask,
    ULONG *mtu_size,
    ULONG *physical_address_msw,
    ULONG *physical_address_lsw);
```

Description

This service retrieves information on network parameters for the specified network interface. All data are retrieved in host byte order.

> **Warning:** *ip_ptr must point to a valid NetX Duo IP structure. The specified interface, if not the primary interface, must be previously attached to the IP instance.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index specifying network interface.
- *interface_name*: Pointer to the buffer that holds the name of the network interface.
- *ip_address*: Pointer to the destination for the IP address of the interface.
- *network_mask*: Pointer to destination for network mask.
- *mtu_size*: Pointer to destination for maximum transfer unit for this interface.
- *physical_address_msw*: Pointer to destination for top 16 bits of the device MAC address.
- *physical_address_lsw*: Pointer to destination for lower 32 bits of the device MAC address.

Return Values

- **NX_SUCCESS** (0x00) Interface information has been obtained.
- **NX_PTR_ERROR** (0x07) Invalid pointer input.
- **NX_INVALID_INTERFACE** (0x4C) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Retrieve interface parameters for the specified interface (index
   1 in IP instance list of interfaces). */
#define INTERFACE_INDEX 1
status = nx_ip_interface_info_get(ip_ptr, INTERFACE_INDEX,
                                  &name_ptr, &ip_address,
                                  &network_mask,
```

```
                                        &mtu_size,
                                        &physical_address_msw,
                                        &physical_address_lsw);

/* If status is NX_SUCCESS the interface information is
   successfully retrieved. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_mtu_set

Set the MTU value of a network interface

Prototype

```
UINT nx_ip_interface_mtu_set(
    NX_IP *ip_ptr,
    UINT interface_index,
    ULONG mtu_size);
```

Description

This service is used by the device driver to configure the IP MTU value for the specified network interface.

Parameters

- *ip_ptr*: IP control block pointer
- *interface_index*: Index to the network interface
- *mtu_size*: IP MTU size

Return Values

- **NX_SUCCESS** (0x00) Successfully set MTU value

- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0
ULONG       mtu_size = 1500;

/* Set the MTU size of specified interface. */
status = nx_ip_interface_mtu_set(&ip_0,
                                 PRIMARY_INTERFACE, mtu_size);

/* If status == NX_SUCCESS, the MTU size was successfully set. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_physical_address_get

Get the physical address of a network device

Prototype

```
UINT nx_ip_interface_physical_address_get(
    NX_IP *ip_ptr,
```

```
    UINT interface_index,
    ULONG *physical_msw,
    ULONG *physical_lsw);
```

Description

This service retrieves the physical address of a network interface from the IP instance.

Parameters

- *ip_ptr*: IP control block pointer
- *interface_index*: Index of the network interface
- *physical_msw*: Pointer to destination for top 16 bits of the device MAC address
- *physical_lsw*: Pointer to destination for lower 32 bits of the device MAC address

Return Values

- **NX_SUCCESS** (0x00) Successful get
- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer or physical address pointer
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0
ULONG   physical_msw;
ULONG   physical_lsw;

/* Get the physical address of specified interface. */
status = nx_ip_interface_physical_address_get(&ip_0,
                                              PRIMARY_INTERFACE,
                                              &physical_msw,
                                              &physical_lsw);
```

```
/* If status == NX_SUCCESS, the physical address was successfully
   retrieved. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_physical_address_set

Set the physical address for a specified network interface

Prototype

```
UINT nx_ip_interface_physical_address_set(
    NX_IP *ip_ptr,
    UINT interface_index,
    ULONG physical_msw,
    ULONG physical_lsw,
    UINT update_driver);
```

Description

This service is used by the application or a device driver to configure the physical address of the MAC address of the specified network interface. The new MAC address is applied to the control block of the interface structure. If the ***update_driver*** flag is set, a driver-level command is issued so the device driver is able to update its MAC address programmed into the Ethernet controller.

In a typical situation, this service is called from the interface device driver during initialization phase to notify the IP stack of its MAC address. In this case, the ***update_driver*** flag should not be set.

This routine can also be called from user application to reconfigure the interface MAC address at run time. In this use case, the ***update_driver*** flag should be set, so the new MAC address can be applied to the device driver.

Parameters

- *ip_ptr*: IP control block pointer
- *interface_index*: Index to the network interface
- *physical_msw*: Pointer to destination for top 16 bits of the device MAC address
- *physical_lsw*: Pointer to destination for lower 32 bits of the device MAC address

Return Values

- **NX_SUCCESS** (0x00) Successful set
- **NX_UNHANDLED_COMMAND** (0x4B) Command not recognized by the driver
- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0
ULONG       physical_msw = 0x00CF;
ULONG       physical_lsw = 0x01020304;

/* Set the physical address of specified device. */
status = nx_ip_interface_physical_address_set(&ip_0,
                                              PRIMARY_INTERFACE,
                                              physical_msw,
                                              physical_lsw,
                                              NX_TRUE);

/* If status == NX_SUCCESS, the physical address was successfully
   set. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set

- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_status_check
- nx_ip_link_status_change_notify_set

nx_ip_interface_status_check

Check status of an IP instance

Prototype

```
UINT nx_ip_interface_status_check(
    NX_IP *ip_ptr,
    UINT interface_index
    ULONG needed_status,
    ULONG *actual_status,
    ULONG wait_option);
```

Description

This service checks and optionally waits for the specified status of the network interface of a previously created IP instance.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Interface index number needed_status IP status requested, defined in bit-map form as follows:
  - *NX_IP_INITIALIZE_DONE** (0x0001)
  - *NX_IP_ADDRESS_RESOLVED** (0x0002)
  - **NX_IP_LINK_ENABLED** (0x0004)
  - **NX_IP_ARP_ENABLED** (0x0008)
  - **NX_IP_UDP_ENABLED** (0x0010)
  - **NX_IP_TCP_ENABLED** (0x0020)
  - **NX_IP_IGMP_ENABLED** (0x0040)
  - **NX_IP_RARP_COMPLETE** (0x0080)
  - **NX_IP_INTERFACE_LINK_ENABLED** (0x0100)
- *actual_status*: Pointer to destination of actual bits set. wait_option Defines how the service behaves if the requested status bits are not available. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **timeout value** (0x00000001 through 0xFFFFFFFE)

– **NX_WAIT_FOREVER** 0xFFFFFFFF

Return Values

- **NX_SUCCESS** (0x00) Successful IP status check.
- **NX_NOT_SUCCESSFUL** (0x43) Status request was not satisfied within the timeout specified.
- **NX_PTR_ERROR** (0x07) IP pointer is or has become invalid, or actual status pointer is invalid.
- **NX_OPTION_ERROR** (0x0a) Invalid needed status option.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_INVALID_INTERFACE** (0x4C) Interface_index is out of range. or the interface is not valid.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Wait 10 ticks for the link up status on the previously created IP
   instance. */
status = nx_ip_interface_status_check(&ip_0, 1, NX_IP_LINK_ENABLED,
                                      &actual_status, 10);

/* If status is NX_SUCCESS, the secondary link for the specified IP
   instance is up. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_link_status_change_notify_set

nx_ip_link_status_change_notify_set

Set the link status change notify callback function

Prototype

```
UINT nx_ip_link_status_change_notify_set(
    NX_IP *ip_ptr,
    VOID(*link_status_change_notify)(NX_IP *ip_ptr, UINT interface_index, UINT link_up));
```

Description

This service configures the link status change notify callback function. The user-supplied *link_status_change_notify* routine is invoked when either the primary or secondary interface status is changed (such as IP address is changed.) If *link_status_change_notify* is NULL, the link status change notify callback feature is disabled.

Parameters

- *ip_ptr*: IP control block pointer
- *link_status_change_notify*: User-supplied callback function to be called upon a change to the physical interface.

Return Values

- **NX_SUCCESS** (0x00) Successful set
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer or new physical address pointer
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Configure a callback function to be used when the physical
   interface status is changed. */
status = nx_ip_link_status_change_notify_set(&ip_0,
                                             my_change_cb);
```

```
/* If status == NX_SUCCESS, the link status change notify function
   is set. */
```

See Also

- nx_ip_interface_address_get
- nx_ip_interface_address_mapping_configure
- nx_ip_interface_address_set
- nx_ip_interface_attach
- nx_ip_interface_capability_get
- nx_ip_interface_capability_set
- nx_ip_interface_detach
- nx_ip_interface_info_get
- nx_ip_interface_mtu_set
- nx_ip_interface_physical_address_get
- nx_ip_interface_physical_address_set
- nx_ip_interface_status_check

nx_ip_max_payload_size_find

Compute maximum packet data payload

Prototype

```
UINT nx_ip_max_payload_size_find(
    NX_IP *ip_ptr,
    NXD_ADDRESS *dest_address,
    UINT if_index,
    UINT src_port,
    UINT dest_port,
    ULONG protocol,
    ULONG *start_offset_ptr,
    ULONG *payload_length_ptr);
```

Description

This service finds the maximum application payload size that will not require IP fragmentation to reach the destination; e.g., payload is at or below the local interface MTU size. (or the Path MTU value obtained via IPv6 Path MTU discovery). IP header and upper application header size (TCP or UDP) are subtracted from the total payload. If NetX Duo IPsec Security Policy applies to this end-point, the IPsec headers (ESP/AH) and associated overhead, such as Initial Vector, are also subtracted from the MTU. This service is applicable for both IPv4 and IPv6 packets.

The parameter *if_index* specifies the interface to use for sending out the packet. For a multihome system, the caller needs to specify the *if_index* parameter if

the destination is a broadcast (IPv4 only), multicast, or IPv6 link-local address.

This service returns two values to the caller:

1) start_offset_ptr: This is the location after the TCP/UDP/IP/IPsec headers;
2) payload_length_ptr: the amount of data application may transfer without exceeding MTU.

There is no equivalent NetX service.

Restrictions

The IP instance must be previously created.

Parameters

- *ip_ptr*: Pointer to IP instance
- *dest_address*: Pointer to packet destination address
- *if_index*: Indicates the index of the interface to use
- *src_port*: Source port number
- *dest_port*: Destination port number
- *protocol*: Upper layer protocol to be used
- *start_offset_ptr*: Pointer to the start of data for maximum packet payload
- *payload_length_ptr*: Pointer to payload size excluding headers

Return Values

- **NX_SUCCESS** (0x00) Payload successfully computed
- **NX_INVALID_INTERFACE** (0x4C) Interface index is invalid, or the interface is not valid.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer, or invalid destination address
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid address supplied
- **NX_NOT_SUPPORTED** (0x4B) Invalid protocol (not UDP or TCP)
- **NX_CALLER_ERROR** (0x11) Service is not called from system initialization or thread context.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* The following example determines the maximum payload for UDP
   packet to remote host. */
#define PRIMARY_INTERFACE 0
status = nx_ip_max_payload_size_find(&ip_0,
                                     &dest_ipv6_address,
                                     PRIMARY_INTERFACE,
                                     source_port,
                                     dest_port, NX_PROTOCOL_UDP,
                                     &start_offset,
                                     &payload_length);

/* A return value of NX_SUCCESS indicates the packet payload
   payload_length starting at the offset start_offset is
   successfully computed. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ip_raw_packet_disable

Disable raw packet sending/receiving

Prototype

```
UINT nx_ip_raw_packet_disable(NX_IP *ip_ptr);
```

Description

This service disables transmission and reception of raw IP packets for this IP instance. If the raw packet service was previously enabled, and there are raw packets in the receive queue, this service will release any received raw packets.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IP raw packet disable.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Disable raw packet sending/receiving for this IP instance. */
status = nx_ip_raw_packet_disable(&ip_0);
/* If status is NX_SUCCESS, raw IP packet sending/receiving has
   been disabled for the previously created IP instance. */
```

See Also

- nx_ip_raw_packet_enable
- nx_ip_raw_packet_filter_set
- nx_ip_raw_packet_receive
- nx_ip_raw_packet_send
- nx_ip_raw_packet_source_send
- nx_ip_raw_receive_queue_max_set
- nxd_ip_raw_packet_send
- nxd_ip_raw_packet_source_send

nx_ip_raw_packet_enable

Enable raw packet processing

Prototype

```
UINT nx_ip_raw_packet_enable(NX_IP *ip_ptr);
```

Description

This service enables transmission and reception of raw IP packets for this IP instance. Incoming TCP, UDP, ICMP, and IGMP packets are still processed by NetX Duo. Packets with unknown upper layer protocol types are processed by raw packet reception routine.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful IP raw packet enable.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Enable raw packet sending/receiving for this IP instance. */
status = nx_ip_raw_packet_enable(&ip_0);

/* If status is NX_SUCCESS, raw IP packet sending/receiving has
   been enabled for the previously created IP instance. */
```

See Also

- nx_ip_raw_packet_disable
- nx_ip_raw_packet_filter_set
- nx_ip_raw_packet_receive
- nx_ip_raw_packet_send
- nx_ip_raw_packet_source_send
- nx_ip_raw_receive_queue_max_set
- nxd_ip_raw_packet_send
- nxd_ip_raw_packet_source_send

nx_ip_raw_packet_filter_set

Set raw IP packet filter

Prototype

```
UINT nx_ip_raw_packet_filter_set(
    NX_IP *ip_ptr,
    UINT (*raw_packet_filter)(NX_IP *, ULONG, NX_PACKET *));
```

Description

This service configures the IP raw packet filter. The raw packet filter function, implemented by user application, allows an application to receive raw packets based on user-supplied criteria. Note that NetX Duo BSD wrapper layer uses the raw packet filter feature to handle raw socket in the BSD layer. To use this service, the NetX Duo library must be built with the option ***NX_ENABLE_IP_RAW_PACKET_FILTER*** defined.

Parameters

- *ip_ptr*: IP control block pointer
- *raw_packet_filter*: Function pointer of the raw packet filter

Return Values

- **NX_SUCCESS** (0x00) Successfully set the raw packet filter routine
- **NX_NOT_SUPPORT** (0x4B) Raw packet support is not available
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
UINT raw_packet_filter(NX_IP *ip_ptr, ULONG protocol,
                       NX_PACKET *packet_ptr)
{

/* Simply filter protocol. */
if(protocol == NX_IP_RAW)
   return NX_SUCCESS;
```

```
else
   return NX_NOT_SUCCESSFUL;
}

void raw_packet_thread_entry(ULONG id)
{

/* Set the raw packet filter of IP instance. */
status = nx_ip_raw_packet_filter_set(&ip_0,
                                     raw_packet_filter);

/* If status == NX_SUCCESS, the raw packet filter of IP instance
   was successfully set. */
}
```

See Also

- nx_ip_raw_packet_disable
- nx_ip_raw_packet_enable
- nx_ip_raw_packet_receive
- nx_ip_raw_packet_send
- nx_ip_raw_packet_source_send
- nx_ip_raw_receive_queue_max_set
- nxd_ip_raw_packet_send
- nxd_ip_raw_packet_source_send

nx_ip_raw_packet_receive

Receive raw IP packet

Prototype

```
UINT nx_ip_raw_packet_receive(
    NX_IP *ip_ptr,
    NX_PACKET **packet_ptr,
    ULONG wait_option);
```

Description

This service receives a raw IP packet from the specified IP instance. If there are IP packets on the raw packet receive queue, the first (oldest) packet is returned to the caller. Otherwise, if no packets are available, the caller may suspend as specified by the wait option.

> **Caution:** *If NX_SUCCESS, is returned, the application is responsible for releasing the received packet when it is no longer needed.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *packet_ptr*: Pointer to pointer to place the received raw IP packet in.
- *wait_option*: Defines how the service behaves if packets are not available. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful IP raw packet receive.
- **NX_NO_PACKET** (0x01) No packet was available.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_PTR_ERROR** (0x07) Invalid IP or return packet pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Threads

Preemption Possible

No

Example

```
/* Receive a raw IP packet for this IP instance, wait for a maximum
   of 4 timer ticks. */
status = nx_ip_raw_packet_receive(&ip_0, &packet_ptr, 4);

/* If status is NX_SUCCESS, the raw IP packet pointer is in the
   variable packet_ptr. */
```

See Also

- nx_ip_raw_packet_disable
- nx_ip_raw_packet_enable
- nx_ip_raw_packet_filter_set
- nx_ip_raw_packet_send
- nx_ip_raw_packet_source_send
- nx_ip_raw_receive_queue_max_set
- nxd_ip_raw_packet_send
- nxd_ip_raw_packet_source_send

nx_ip_raw_packet_send

Send raw IP packet

Prototype

```
UINT nx_ip_raw_packet_send(
    NX_IP *ip_ptr,
    NX_PACKET *packet_ptr,
    ULONG destination_ip,
    ULONG type_of_service);
```

Description

This service sends a raw IPv4 packet to the destination IP address. Note that this routine returns immediately, and it is therefore not known whether the IP packet has actually been sent. The network driver will be responsible for releasing the packet when the transmission is complete.

For a multihome system, NetX Duo uses the destination IP address to find an appropriate network interface and uses the IP address of the interface as the source address. If the destination IP address is broadcast or multicast, the first valid interface is used. Applications use the ***nx_ip_raw_packet_source_send*** in this case.

To send a raw IPv6 packet, application shall use the service ***nxd_ip_raw_packet_send,*** or ***nxd_ip_raw_packet_source_send.***

> **Warning:** *Unless an error is returned, the application should not release the packet after this call. Doing so will cause unpredictable results because the network driver will release the packet after transmission.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *packet_ptr*: Pointer to the raw IP packet to send.
- *destination_ip*: Destination IP address, which can be a specific host IP address, a network broadcast, an internal loop-back, or a multicast address.
- *type_of_service*: Defines the type of service for the transmission, legal values are as follows:
  - **NX_IP_NORMAL** (0x00000000)
  - **NX_IP_MIN_DELAY** (0x00100000)
  - **NX_IP_MAX_DATA** (0x00080000)
  - **NX_IP_MAX_RELIABLE** (0x00040000)
  - **NX_IP_MIN_COST** (0x00020000)

Return Values

- **NX_SUCCESS** (0x00) Successful IP raw packet send initiated.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_NOT_ENABLED** (0x14) Raw IP feature is not enabled.
- **NX_OPTION_ERROR** (0x0A) Invalid type of service.
- **NX_UNDERFLOW** (0x02) Not enough room to prepend an IP header on the packet.
- **NX_OVERFLOW** (0x03) Packet append pointer is invalid.
- **NX_PTR_ERROR** (0x07) Invalid IP or packet pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Send a raw IP packet to IP address 1.2.3.5. */
status = nx_ip_raw_packet_send(&ip_0, packet_ptr,
                               IP_ADDRESS(1,2,3,5),
                               NX_IP_NORMAL);

/* If status is NX_SUCCESS, the raw IP packet pointed to by
   packet_ptr has been sent. */
```

See Also

- nx_ip_raw_packet_disable
- nx_ip_raw_packet_enable
- nx_ip_raw_packet_filter_set
- nx_ip_raw_packet_receive
- nx_ip_raw_packet_send
- nx_ip_raw_packet_source_send
- nx_ip_raw_receive_queue_max_set
- nxd_ip_raw_packet_send
- nxd_ip_raw_packet_source_send

nx_ip_raw_packet_source_send

Send raw IP packet through specified network interface

Prototype

```
UINT nx_ip_raw_packet_source_send(
    NX_IP *ip_ptr,
    NX_PACKET *packet_ptr,
    ULONG destination_ip,
    UINT address_index,
    ULONG type_of_service);
```

Description

This service sends a raw IP packet to the destination IP address using the specified local IPv4 address as the source address, and through the associated network interface. Note that this routine returns immediately, and it is, therefore, not known if the IP packet has actually been sent. The network driver will be responsible for releasing the packet when the transmission is complete. This service differs from other services in that there is no way of knowing if the packet was actually sent. It could get lost on the Internet.

> **Caution:** *Note that raw IP processing must be enabled.*
>
> **Warning:** *This service is similar to* ***nx_ip_raw_packet_send,*** *except that this service allows an application to send raw IPv4 packet from a specified physical interfaces.*

Parameters

- *ip_ptr*: Pointer to previously created IP task.
- *packet_ptr*: Pointer to packet to transmit.
- *destination_ip*: IP address to send packet.
- *address_index*: Index of the address of the interface to send packet out on.
- *type_of_service*: Type of service for packet.

Return Values

- **NX_SUCCESS** (0x00) Packet successfully transmitted.
- **NX_IP_ADDRESS_ERROR** (0x21) No suitable outgoing interface available.
- **NX_NOT_ENABLED** (0x14) Raw IP packet processing not enabled.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid pointer input.
- **NX_OPTION_ERROR** (0x0A) Invalid type of service specified.
- **NX_OVERFLOW** (0x03) Invalid packet prepend pointer.
- **NX_UNDERFLOW** (0x02) Invalid packet prepend pointer.
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index specified.

Allowed From

Threads

Preemption Possible

No

Example

```
#define ADDRESS_IDNEX 1

/* Send packet out on interface 1 with normal type of service. */
status = nx_ip_raw_packet_source_send(ip_ptr, packet_ptr,
                                      destination_ip,
                                      ADDRESS_INDEX,
                                      NX_IP_NORMAL);

/* If status is NX_SUCCESS the packet was successfully
   transmitted. */
```

See Also

- nx_ip_raw_packet_disable
- nx_ip_raw_packet_enable
- nx_ip_raw_packet_filter_set
- nx_ip_raw_packet_receive
- nx_ip_raw_packet_send
- nx_ip_raw_receive_queue_max_set
- nxd_ip_raw_packet_send
- nxd_ip_raw_packet_source_send

nx_ip_raw_receive_queue_max_set

Set maximum raw receive queue size

Prototype

```
UINT nx_ip_raw_receive_queue_max_set(
    NX_IP *ip_ptr,
    ULONG queue_max);
```

Description

This service configures the maximum depth of the IP raw packet receive queue. Note that the IP raw packet receive queue is shared with both IPv4 and IPv6 packets. When the raw packet receive queue reaches the userconfigured maximum

depth, newly received raw packets are dropped. The default IP raw packet receive queue depth is 20.

Parameters

- *ip_ptr*: IP control block pointer
- *queue_max*: New value for the queue size

Return Values

- **NX_SUCCESS** (0x00) Successfully set raw receive queue maximum depth
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization and threads

Preemption Possible

No

Example

```
ULONG queue_max = 10;

/* Set the maximum size of the IP raw packet receive queue. */
status = nx_ip_raw_receive_queue_max_set (&ip_0,
                                          queue_max);

/* If status == NX_SUCCESS, the maximum size of the IP raw packet
   receive queue was successfully set. */
```

See Also

- nx_ip_raw_packet_disable
- nx_ip_raw_packet_enable
- nx_ip_raw_packet_filter_set
- nx_ip_raw_packet_receive
- nx_ip_raw_packet_send
- nx_ip_raw_packet_source_send
- nxd_ip_raw_packet_send
- nxd_ip_raw_packet_source_send

nx_ip_static_route_add

Add static route to the routing table

Prototype

```
UINT nx_ip_static_route_add(
    NX_IP *ip_ptr,
    ULONG network_address,
    ULONG net_mask,
    ULONG next_hop);
```

Description

This service adds an entry to the static routing table. Note that the *next_hop* address must be directly accessible from one of the local network devices.

> **Caution:** *Note that ip_ptr must point to a valid NetX Duo IP structure and the NetX Duo library must be built with NX_ENABLE_IP_STATIC_ROUTING defined to use this service. By default NetX Duo is built without NX_ENABLE_IP_STATIC_ROUTING defined.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *network_address*: Target network address, in host byte order
- *net_mask*: Target network mask, in host byte order
- *next_hop*: Next hop address for the target network, in host byte order

Return Values

- **NX_SUCCESS** (0x00) Entry is added to the static routing table.
- **NX_OVERFLOW** (0x03) Static routing table is full.
- **NX_NOT_SUPPORTED** (0x4B) This feature is not compiled in.
- **NX_IP_ADDRESS_ERROR** (0x21) Next hop is not directly accessible via local interfaces.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid ip_ptr pointer.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Specify the next hop for 192.168.1.68 through the gateway
   192.168.1.1. */
```

```
status = nx_ip_static_route_add(ip_ptr, IP_ADDRESS(192,168,1,0),
                                0xFFFFFF00UL,
                                IP_ADDRESS(192,168,1,1));

/* If status is NX_SUCCESS the route was successfully added to the
   static routing table. */
```

See Also

- nx_ip_gateway_address_clear
- nx_ip_gateway_address_get
- nx_ip_gateway_address_set
- nx_ip_info_get
- nx_ip_static_route_delete
- nxd_ipv6_default_router_add
- nxd_ipv6_default_router_delete
- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_get
- nxd_ipv6_default_router_number_of_entries_get

nx_ip_static_route_delete

Delete static route from routing table

Prototype

```
UINT nx_ip_static_route_delete(
    NX_IP *ip_ptr,
    ULONG network_address,
    ULONG net_mask);
```

Description

This service deletes an entry from the static routing table.

> **Warning:** *Note that ip_ptr must point to a valid NetX Duo IP structure and the NetX Duo library must be built with NX_ENABLE_IP_STATIC_ROUTING defined to use this service. By default NetX Duo is built without NX_ENABLE_IP_STATIC_ROUTING defined.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *network_address*: Target network address, in host byte order.
- *net_mask*: Target network mask, in host byte order.

Return Values

- **NX_SUCCESS** (0x00) Successful deletion from the static routing table.
- **NX_NOT_SUCCESSFUL** (0x43) Entry cannot be found in the routing table.
- **NX_NOT_SUPPORTED** (0x4B) This feature is not compiled in.
- **NX_PTR_ERROR** (0x07) Invalid ip_ptr pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Remove the static route for 192.168.1.68 from the routing
   table.*/
status = nx_ip_static_route_delete(ip_ptr,
                                   IP_ADDRESS(192,168,1,0),
                                   0xFFFFFF00UL,);

/* If status is NX_SUCCESS the route was successfully removed from
   the static routing table. */
```

See Also

- nx_ip_gateway_address_clear
- nx_ip_gateway_address_get
- nx_ip_gateway_address_set
- nx_ip_info_get
- nx_ip_static_route_add
- nxd_ipv6_default_router_add
- nxd_ipv6_default_router_delete
- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_get
- nxd_ipv6_default_router_number_of_entries_get

nx_ip_status_check

Check status of an IP instance

Prototype

```
UINT nx_ip_status_check(
    NX_IP *ip_ptr,
    ULONG needed_status,
    ULONG *actual_status,
    ULONG wait_option);
```

Description

This service checks and optionally waits for the specified status of the primary network interface of a previously created IP instance. To obtain status on secondary interfaces, applications shall use the service ***nx_ip_interface_status_check.***

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *needed_status*: IP status requested, defined in bit-map form as follows:
  - **NX_IP_INITIALIZE_DONE** (0x0001)
  - **NX_IP_ADDRESS_RESOLVED** (0x0002)
  - **NX_IP_LINK_ENABLED** (0x0004)
  - **NX_IP_ARP_ENABLED** (0x0008)
  - **NX_IP_UDP_ENABLED** (0x0010)
  - **NX_IP_TCP_ENABLED** (0x0020)
  - **NX_IP_IGMP_ENABLED** (0x0040)
  - **NX_IP_RARP_COMPLETE** (0x0080)
  - **NX_IP_INTERFACE_LINK_ENABLED** (0x0100)
- *actual_status*: Pointer to destination of actual bits set.
- *wait_option*: Defines how the service behaves if the requested status bits are not available. The wait options are defined as follows:
  - **NX_NO_WAIT** 0x00000000)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)
  - **NX_WAIT_FOREVER** 0xFFFFFFFF

Return Values

- **NX_SUCCESS** (0x00) Successful IP status check.
- **NX_NOT_SUCCESSFUL** (0x43) Status request was not satisfied within the timeout specified.
- **NX_PTR_ERROR** (0x07) IP pointer is or has become invalid, or actual status pointer is invalid.
- **NX_OPTION_ERROR** (0x0a) Invalid needed status option.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Wait 10 ticks for the link up status on the previously created IP
   instance. */
status = nx_ip_status_check(&ip_0, NX_IP_LINK_ENABLED,
                            &actual_status, 10);

/* If status is NX_SUCCESS, the link for the specified IP instance
   is up. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_ipv4_multicast_interface_join

Join IP instance to specified multicast group via an interface

Prototype

```
UINT nx_ipv4_multicast_interface_join(
    NX_IP *ip_ptr,
    ULONG group_address,
    UINT interface_index);
```

Description

This service joins an IP instance to the specified multicast group via a specified network interface. Once the IP instance joins a multicast group, the IP receive logic starts to forward data packets from the give multicast group to the upper layer. Note that this service joins a multicast group without sending IGMP reports.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *group_address*: Class D IP multicast group address to join in host byte order.
- *interface_index*: Index of the Interface attached to the NetX Duo instance.

Return Values

- **NX_SUCCESS** (0x00) Successful multicast group join.
- **NX_NO_MORE_ENTRIES** (0x17) No more multicast groups can be joined, maximum exceeded.
- **NX_PTR_ERROR** (0x07) Invalid pointer to IP instance, or the IP instance is invalid
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_EANABLED** (0x14) IGMP is not enabled in this IP instance
- **NX_IP_ADDRESS_ERROR** (0x21) Multicast group address provided is not a valid class D address.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Previously created IP Instance joins the multicast group
   224.0.0.200, via the interface at index 1 in the IP interface
   list. */
#define INTERFACE_INDEX 1
status = nx_ipv4_multicast_interface_join
                                  (&ip IP_ADDRESS(224,0,0,200),
                                   INTERFACE_INDEX);

/* If status is NX_SUCCESS, the IP instance has successfully joined
   the multicast group. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_getnx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_ipv4_multicast_interface_leave

Leave specified multicast group via an interface

Prototype

```
UINT nx_ipv4_multicast_interface_leave(
    NX_IP *ip_ptr,
    ULONG group_address,
    UINT interface_index);
```

Description

This service leaves the specified multicast group via a specified network interface. After leaving the group, this service does not trigger IGMP messages being generated.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *group_address*: Class D IP multicast group address to leave. The IP address is in host byte order.

- *interface_index*: Index of the Interface attached to the NetX Duo instance.

Return Values

- **NX_SUCCESS** (0x00) Successful multicast group join.
- **NX_ENTRY_NOT_FOUND** (0x16) The specified multicast group address cannot be found in the local multicast table.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.
- **NX_IP_ADDRESS_ERROR** (0x21) Multicast group address provided is not a valid class D address.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid pointer to IP instance, or the IP instance is invalid

Allowed From

Threads

Preemption Possible

No

Example

```
/* Leave the multicast group 224.0.0.200. */
#define INTERFACE_INDEX 1
status = nx_ipv4_multicast_interface_leave
                                (&ip, IP_ADDRESS(224,0,0,200),
                                 INTERFACE_INDEX);

/* If status is NX_SUCCESS, the IP instance has successfully leaves
   the multicast group 244.0.0.200. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_get
- nx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join
- nxd_ipv6_multicast_interface_join
- nxd_ipv6_multicast_interface_leave

nx_packet_allocate

Allocate packet from specified pool

Prototype

```
UINT nx_packet_allocate(
    NX_PACKET_POOL *pool_ptr,
    NX_PACKET **packet_ptr,
    ULONG packet_type,
    ULONG wait_option);
```

Description

This service allocates a packet from the specified pool and adjusts the prepend pointer in the packet according to the type of packet specified. If no packet is available, the service suspends according to the supplied wait option.

Parameters

- *pool_ptr*: Pointer to previously created packet pool.
- *packet_ptr*: Pointer to the Pointer of the allocated packet Pointer.
- *packet_type*: Defines the type of packet requested. See "Packet Pools" on page 63 in Chapter 3 for a list of supported packet types.
- *wait_option*: Defines the wait time in ticks if there are no packets available in the packet pool. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful packet allocate.
- **NX_NO_PACKET** (0x01) No packet available.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_INVALID_PARAMETERS** (0x4D) Packet size cannot support protocol.
- **NX_OPTION_ERROR** (0x0A) Invalid packet type.
- **NX_PTR_ERROR** (0x07) Invalid pool or packet return pointer.
- **NX_CALLER_ERROR** (0x11) Invalid wait option from nonthread.

Allowed From

Initialization, threads, timers, and ISRs (application network drivers). Wait option must be *NX_NO_WAIT* when used in ISR or in timer context.

Preemption Possible

No

Example

```
/* Allocate a new UDP packet from the previously created packet pool
   and suspend for a maximum of 5 timer ticks if the pool is
   empty. */
status = nx_packet_allocate(&pool_0, &packet_ptr,
                            NX_UDP_PACKET, 5);

/* If status is NX_SUCCESS, the newly allocated packet pointer is
   found in the variable packet_ptr. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_packet_copy

Copy packet

Prototype

```
UINT nx_packet_copy(
    NX_PACKET *packet_ptr,
    NX_PACKET **new_packet_ptr,
    NX_PACKET_POOL *pool_ptr,
    ULONG wait_option);
```

Description

This service copies the information in the supplied packet to one or more new packets that are allocated from the supplied packet pool. If successful, the pointer to the new packet is returned in destination pointed to by **new_packet_ptr**.

Parameters

- *packet_ptr*: Pointer to the source packet.
- *new_packet_ptr*: Pointer to the destination of where to return the pointer to the new copy of the packet.
- *pool_ptr*: Pointer to the previously created packet pool that is used to allocate one or more packets for the copy.
- *wait_option*: Defines how the service waits if there are no packets available. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful packet copy.
- **NX_NO_PACKET** (0x01) Packet not available for copy.
- **NX_INVALID_PACKET** (0x12) Empty source packet or copy failed.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_INVALID_PARAMETERS** (0x4D) Packet size cannot support protocol.
- **NX_PTR_ERROR** (0x07) Invalid pool, packet, or destination pointer.
- **NX_UNDERFLOW** (0x02) Invalid packet prepend pointer.
- **NX_OVERFLOW** (0x03) Invalid packet append pointer.
- **NX_CALLER_ERROR** (0x11) A wait option was specified in initialization or in an ISR.

Allowed From

Initialization, threads, timers, and ISRs

Preemption Possible

No

Example

```
NX_PACKET *new_copy_ptr;

/* Copy packet pointed to by "old_packet_ptr" using packets from
   previously created packet pool_0. */
status = nx_packet_copy(old_packet, &new_copy_ptr, &pool_0, 20);

/* If status is NX_SUCCESS, new_copy_ptr points to the packet copy. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_packet_data_append

Append data to end of packet

Prototype

```
UINT nx_packet_data_append(
    NX_PACKET *packet_ptr,
    VOID *data_start, ULONG data_size,
    NX_PACKET_POOL *pool_ptr,
    ULONG wait_option);
```

Description

This service appends data to the end of the specified packet. The supplied data area is copied into the packet. If there is not enough memory available, and the chained packet feature is enabled, one or more packets will be allocated to satisfy the request. If the chained packet feature is not enabled, *NX_SIZE_ERROR* is returned.

Parameters

- *packet_ptr*: Packet pointer.
- *data_start*: Pointer to the start of the user's data area to append to the packet.
- *data_size*: Size of user's data area.
- *pool_ptr*: Pointer to packet pool from which to allocate another packet if there is not enough room in the current packet.
- *wait_option*: Defines how the service behaves if there are no packets available. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)

– **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful packet append.
- **NX_NO_PACKET** (0x01) No packet available.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_INVALID_PARAMETERS** (0x4D) Packet size cannot support protocol.
- **NX_UNDERFLOW** (0x02) Prepend pointer is less than payload start.
- **NX_OVERFLOW** (0x03) Append pointer is greater than payload end.
- **NX_PTR_ERROR** (0x07) Invalid pool, packet, or data Pointer.
- **NX_SIZE_ERROR** (0x09) Invalid data size.
- **NX_CALLER_ERROR** (0x11) Invalid wait option from nonthread.

Allowed From

Initialization, threads, timers, and ISRs (application network drivers)

Preemption Possible

No

Example

```
/* Append "abcd" to the specified packet. */
status = nx_packet_data_append(packet_ptr, "abcd", 4, &pool_0, 5);

/* If status is NX_SUCCESS, the additional four bytes "abcd" have
   been appended to the packet. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_packet_data_extract_offset

Extract data from packet via an offset

Prototype

```
UINT nx_packet_data_extract_offset(
    NX_PACKET *packet_ptr,
    ULONG offset,
    VOID *buffer_start,
    ULONG buffer_length,
    ULONG *bytes_copied);
```

Description

This service copies data from a NetX Duo packet (or packet chain) starting at the specified offset from the packet prepend pointer of the specified size in bytes into the specified buffer. The number of bytes actually copied is returned in *bytes_copied.* This service does not remove data from the packet, nor does it adjust the prepend pointer or other internal state information.

Parameters

- *packet_ptr*: Pointer to packet to extract
- *offset*: Offset from the current prepend pointer.
- *buffer_start*: Pointer to start of save buffer
- *buffer_length*: Number of bytes to copy
- *bytes_copied*: Number of bytes actually copied

Return Values

- **NX_SUCCESS** (0x00) Successful packet copy
- **NX_PACKET_OFFSET_ERROR** (0x53) Invalid offset value was supplied
- **NX_PTR_ERROR** (0x07) Invalid packet pointer or buffer pointer

Allowed From

Initialization, threads, timers, and ISRs

Preemption Possible

No

```
/* Extract 10 bytes from the start of the received packet buffer
   into the specified memory area. */
status = nx_packet_data_extract_offset(my_packet, 0, &data[0], 10,
                                       &bytes_copied) ;
```

```
/* If status is NX_SUCCESS, 10 bytes were successfully copied into
   the data buffer. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_packet_data_retrieve

Retrieve data from packet

Prototype

```
UINT nx_packet_data_retrieve(
    NX_PACKET *packet_ptr,
    VOID *buffer_start,
    ULONG *bytes_copied);
```

Description

This service copies data from the supplied packet into the supplied buffer. The actual number of bytes copied is returned in the destination pointed to by **bytes_copied**.

Note that this service does not change internal state of the packet. The data being retrieved is still available in the packet.

> **Caution:** *The destination buffer must be large enough to hold the packet's contents. If not, memory will be corrupted causing unpredictable results.*

Parameters

- *packet_ptr*: Pointer to the source packet.
- *buffer_start*: Pointer to the start of the buffer area.
- *bytes_copied*: Pointer to the destination for the number of bytes copied.

Return Values

- **NX_SUCCESS** (0x00) Successful packet data retrieve.
- **NX_INVALID_PACKET** (0x12) Invalid packet.
- **NX_PTR_ERROR** (0x07) Invalid packet, buffer start, or bytes copied pointer.

Allowed From

Initialization, threads, timers, and ISRs

Preemption Possible

No

Example

```
UCHAR                 buffer[512];
ULONG                 bytes_copied;

/* Retrieve data from packet pointed to by "packet_ptr". */
status = nx_packet_data_retrieve(packet_ptr, buffer, &bytes_copied);

/* If status is NX_SUCCESS, buffer contains the contents of the
   packet, the size of which is contained in "bytes_copied." */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_packet_length_get

Get length of packet data

Prototype

```
UINT nx_packet_length_get(
    NX_PACKET *packet_ptr,
    ULONG *length);
```

Description

This service gets the length of the data in the specified packet.

Parameters

- *packet_ptr*: Pointer to the packet.
- *length*: Destination for the packet length.

Return Values

- **NX_SUCCESS** (0x00) Successful packet length get.
- **NX_PTR_ERROR** (0x07) Invalid packet pointer.

Allowed From

Initialization, threads, timers, and ISRs

Preemption Possible

No

Example

```
/* Get the length of the data in "my_packet." */
status = nx_packet_length_get(my_packet, &my_length);

/* If status is NX_SUCCESS, data length is in "my_length". */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release

- nx_packet_transmit_release

nx_packet_pool_create

Create packet pool in specified memory area

Prototype

```
UINT nx_packet_pool_create(
    NX_PACKET_POOL *pool_ptr,
    CHAR *name,
    ULONG payload_size,
    VOID *memory_ptr,
    ULONG memory_size);
```

Description

This service creates a packet pool of the specified packet size in the memory area supplied by the user.

Parameters

- *pool_ptr*: Pointer to packet pool control block.
- *name*: Pointer to application's name for the packet pool.
- *payload_size*: Number of bytes in each packet in the pool. This value must be at least 40 bytes and must also be evenly divisible by 4.
- *memory_ptr*: Pointer to the memory area to place the packet pool in. The pointer should be aligned on an ULONG boundary.
- *memory_size*: Size of the pool memory area.

Return Values

- **NX_SUCCESS** (0x00) Successful packet pool create.
- **NX_PTR_ERROR** (0x07) Invalid pool or memory pointer.
- **NX_SIZE_ERROR** (0x09) Invalid block or memory size.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Create a packet pool of 32000 bytes starting at physical
   address 0x10000000. */
status = nx_packet_pool_create(&pool_0, "Default Pool", 128,
                               (void *) 0x10000000, 32000);

/* If status is NX_SUCCESS, the packet pool has been successfully
   created. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_packet_pool_delete

Delete previously created packet pool

Prototype

```
UINT  nx_packet_pool_delete(NX_PACKET_POOL *pool_ptr);
```

Description

This service deletes a previously created packet pool. NetX Duo checks for any threads currently suspended on packets in the packet pool and clears the suspension.

Parameters

- *pool_ptr*: Packet pool control block pointer.

Return Values

- **NX_SUCCESS** (0x00) Successful packet pool delete.
- **NX_PTR_ERROR** (0x07) Invalid pool pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

Yes

Example

```
/* Delete a previously created packet pool. */
status = nx_packet_pool_delete(&pool_0);

/* If status is NX_SUCCESS, the packet pool has been successfully
   deleted. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_packet_pool_info_get

Retrieve information about a packet pool

Prototype

```
UINT 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);
```

Description

This service retrieves information about the specified packet pool.

Important: *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *pool_ptr*: Pointer to previously created packet pool.
- *total_packets*: Pointer to destination for the total number of packets in the pool.
- *free_packets*: Pointer to destination for the total number of currently free packets.
- *empty_pool_requests*: Pointer to destination of the total number of allocation requests when the pool was empty.
- *empty_pool_suspensions*: Pointer to destination of the total number of empty pool suspensions.
- *invalid_packet_releases*: Pointer to destination of the total number of invalid packet releases.

Return Values

- **NX_SUCCESS** (0x00) Successful packet pool information retrieval.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads, and timers

Preemption Possible

No

Example

```
/* Retrieve packet pool information. */
status = nx_packet_pool_info_get(&pool_0,
                                 &total_packets,
                                 &free_packets,
                                 &empty_pool_requests,
                                 &empty_pool_suspensions,
                                 &invalid_packet_releases);

/* If status is NX_SUCCESS, packet pool information was
   retrieved. */
```

See Also

- nx_ip_auxiliary_packet_pool_set

- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_low_watermark_set
- nx_packet_release
- nx_packet_transmit_release

nx_packet_pool_low_watermark_set

Set packet pool low watermark

Prototype

```
UINT nx_packet_pool_low_watermark_set(
    NX_PACKET_POOL *pool_ptr,
    ULONG low_watermark);
```

Description

This service configures the low watermark for the specified packet pool. Once the low watermark value is set, TCP or UDP will not queue up the received packets if the number of available packets in the packet pool is less than the packet pool's low watermark, preventing the packet pool from being starved of packets. This service is available if the NetX Duo library is built with the option **NX_ENABLE_LOW_WATERMARK** defined.

Parameters

- *pool_ptr*: Pointer to packet pool control block.
- *low_watermark*: Low watermark value to be configured

Return Values

- **NX_SUCCESS** (0x00) Successfully set the low watermark value.
- **NX_NOT_SUPPORTED** (0x4B) The low watermark feature is not built into NetX Duo.
- **NX_PTR_ERROR** (0x07) Invalid pool pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Set pool_0 low watermark value to 2. */
status = nx_packet_pool_create(&pool_0, 2);

/* If status is NX_SUCCESS, the low watermark value is set for
   pool_0.*/
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_release
- nx_packet_transmit_release

nx_packet_release

Release previously allocated packet

Prototype

```
UINT nx_packet_release(NX_PACKET *packet_ptr);
```

Description

This service releases a packet, including any additional packets chained to the specified packet. If another thread is blocked on packet allocation, it is given the packet and resumed.

> [!NOTE]
> *The application must prevent releasing a packet more than once, because doing so will cause unpredictable results.*

Parameters

- *packet_ptr*: Packet pointer.

Return Values

- **NX_SUCCESS** (0x00) Successful packet release.
- **NX_PTR_ERROR** (0x07) Invalid packet pointer.
- **NX_UNDERFLOW** (0x02) Prepend pointer is less than payload start.
- **NX_OVERFLOW** (0x03) Append pointer is greater than payload end.

Allowed From

Initialization, threads, timers, and ISRs (application network drivers)

Preemption Possible

Yes

Example

```
/* Release a previously allocated packet. */
status = nx_packet_release(packet_ptr);

/* If status is NX_SUCCESS, the packet has been returned to the
   packet pool it was allocated from. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve
- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_transmit_release

nx_packet_transmit_release

Release a transmitted packet

Prototype

```
UINT nx_packet_transmit_release(NX_PACKET *packet_ptr);
```

Description

For non-TCP packets, this service releases a transmitted packet, including any additional packets chained to the specified packet. If another thread is blocked on packet allocation, it is given the packet and resumed. For a transmitted TCP packet, the packet is marked as being transmitted but not released till the packet is acknowledged. This service is typically called from the application's network driver after a packet is transmitted.

Warning: *The network driver should remove the physical media header and adjust the length of the packet before calling this service.*

Parameters

- *packet_ptr*: Packet pointer.

Return Values

- **NX_SUCCESS** (0x00) Successful transmit packet release.
- **NX_PTR_ERROR** (0x07) Invalid packet pointer.
- **NX_UNDERFLOW** (0x02) Prepend pointer is less than payload start.
- **NX_OVERFLOW** (0x03) Append pointer is greater than payload end.

Allowed From

Initialization, threads, timers, Application network drivers (including ISRs)

Preemption Possible

Yes

Example

```
/* Release a previously allocated packet that was just transmitted
   from the application network driver. */
status = nx_packet_transmit_release(packet_ptr);

/* If status is NX_SUCCESS, the transmitted packet has been
   returned to the packet pool it was allocated from. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_packet_allocate
- nx_packet_copy
- nx_packet_data_append
- nx_packet_data_extract_offset
- nx_packet_data_retrieve

- nx_packet_length_get
- nx_packet_pool_create
- nx_packet_pool_delete
- nx_packet_pool_info_get
- nx_packet_pool_low_watermark_set
- nx_packet_release

nx_rarp_disable

Disable Reverse Address Resolution Protocol (RARP)

Prototype

```
UINT nx_rarp_disable(NX_IP *ip_ptr);
```

Description

This service disables the RARP component of NetX Duo for the specific IP instance. For a multihome system, this service disables RARP on all interfaces.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful RARP disable.
- **NX_NOT_ENABLED** (0x14) RARP was not enabled.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Disable RARP on the previously created IP instance. */
status = nx_rarp_disable(&ip_0);

/* If status is NX_SUCCESS, RARP is disabled. */
```

See Also

- nx_rarp_enable
- nx_rarp_info_get

nx_rarp_enable

Enable Reverse Address Resolution Protocol (RARP)

Prototype

```
UINT nx_rarp_enable(NX_IP *ip_ptr);
```

Description

This service enables the RARP component of NetX Duo for the specific IP instance. The RARP components searches through all attached network interfaces for zero IP address. A zero IP address indicates the interface does not have IP address assignment yet. RARP attempts to resolve the IP address by enabling RARP process on that interface.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful RARP enable.
- **NX_IP_ADDRESS_ERROR** (0x21) IP address is already valid.
- **NX_ALREADY_ENABLED** (0x15) RARP was already enabled.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads, timers

Preemption Possible

No

Example

```
/* Enable RARP on the previously created IP instance. */
status = nx_rarp_enable(&ip_0);

/* If status is NX_SUCCESS, RARP is enabled and is attempting to
   resolve this IP instance's address by querying the network. */
```

See Also

- nx_rarp_disable
- nx_rarp_info_get

nx_rarp_info_get

Retrieve information about RARP activities

Prototype

```
UINT nx_rarp_info_get(
    NX_IP *ip_ptr,
    ULONG *rarp_requests_sent,
    ULONG *rarp_responses_received,
    ULONG *rarp_invalid_messages);
```

Description

This service retrieves information about RARP activities for the specified IP instance.

> **Important:** *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *rarp_requests_sent*: Pointer to destination for the total number of RARP requests sent.
- *rarp_responses_received*: Pointer to destination for the total number of RARP responses received.
- *rarp_invalid_messages*: Pointer to destination of the total number of invalid messages.

Return Values

- **NX_SUCCESS** (0x00) Successful RARP information retrieval.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Retrieve RARP information from previously created IP
   Instance 0. */
status = nx_rarp_info_get(&ip_0,
                          &rarp_requests_sent,
                          &rarp_responses_received,
                          &rarp_invalid_messages);

/* If status is NX_SUCCESS, RARP information was retrieved. */
```

See Also

- nx_rarp_disable
- nx_rarp_enable

nx_system_initialize

Initialize NetX Duo System

Prototype

```
VOID nx_system_initialize(VOID);
```

Description

This service initializes the basic NetX Duo system resources in preparation for use. It should be called by the application during initialization and before any other NetX Duo call are made.

Parameters

None

Return Values

None

Allowed From

Initialization, threads, timers, ISRs

Preemption Possible

No

System Management

Example

```
/* Initialize NetX Duo for operation. */
nx_system_initialize();

/* At this point, NetX Duo is ready for IP creation and all
   subsequent network operations. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nx_tcp_client_socket_bind

Bind client TCP socket to TCP port

Prototype

```
UINT nx_tcp_client_socket_bind(
    NX_TCP_SOCKET *socket_ptr,
    UINT port,
    ULONG wait_option);
```

Description

This service binds the previously created TCP client socket to the specified TCP port. Valid TCP sockets range from 0 through 0xFFFF. If the specified TCP port is unavailable, the service suspends according to the supplied wait option.

Parameters

- *socket_ptr*: Pointer to previously created TCP socket instance.
- *port Port*: number to bind (1 through 0xFFFF). If port number is NX_ANY_PORT (0x0000), the IP instance will search for the next free port and use that for the binding.
- *wait_option*: Defines how the service behaves if the port is already bound to another socket. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
- *timeout value in ticks*: (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful socket bind.
- **NX_ALREADY_BOUND** (0x22) This socket is already bound to another TCP port.
- **NX_PORT_UNAVAILABLE** (0x23) Port is already bound to a different socket.
- **NX_NO_FREE_PORTS** (0x45) No free port.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_INVALID_PORT** (0x46) Invalid port.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Bind a previously created client socket to port 12 and wait for 7
   timer ticks for the bind to complete. */
status = nx_tcp_client_socket_bind(&client_socket, 12, 7);
```

```
/* If status is NX_SUCCESS, the previously created client_socket is
   bound to port 12 on the associated IP instance. */
```

See Also

- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_client_socket_connect

Connect client TCP socket

Prototype

```
UINT nx_tcp_client_socket_connect(
    NX_TCP_SOCKET *socket_ptr,
    ULONG server_ip,
    UINT server_port,
    ULONG wait_option);
```

Description

This service connects the previously created and bound TCP client socket to the specified server's port. Valid TCP server ports range from 0 through 0xFFFF. If the connection does not complete immediately, the service suspends according to the supplied wait option.

Parameters

- *socket_ptr*: Pointer to previously created TCP socket instance.
- *server_ip*: Server's IP address.
- *server_port*: Server port number to connect to (1 through 0xFFFF).
- *wait_option*: Defines how the service behaves while the connection is being established. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful socket connect.
- **NX_NOT_BOUND** (0x24) Socket is not bound.
- **NX_NOT_CLOSED** (0x35) Socket is not in a closed state.
- **NX_IN_PROGRESS** (0x37) No wait was specified, the connection attempt is in progress.
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface supplied.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid server IP address.
- **NX_INVALID_PORT** (0x46) Invalid port.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Initiate a TCP connection from a previously created and bound
   client socket. The connection requested in this example is to
   port 12 on the server with the IP address of 1.2.3.5. This
   service will wait 300 timer ticks for the connection to take
   place before giving up. */
status = nx_tcp_client_socket_connect(&client_socket,
                                      IP_ADDRESS(1,2,3,5),
                                      12, 300);
```

```
/* If status is NX_SUCCESS, the previously created and bound
   client_socket is connected to port 12 on IP 1.2.3.5. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_client_socket_port_get

Get port number bound to client TCP socket

Prototype

```
UINT nx_tcp_client_socket_port_get(
    NX_TCP_SOCKET *socket_ptr,
    UINT *port_ptr);
```

Description

This service retrieves the port number associated with the socket, which is useful to find the port allocated by NetX Duo in situations where the NX_ANY_PORT was specified at the time the socket was bound.

Parameters

- *socket_ptr*: Pointer to previously created TCP socket instance.

- *port_ptr*: Pointer to destination for the return port number. Valid port numbers are (1 through 0xFFFF).

Return Values

- **NX_SUCCESS** (0x00) Successful socket bind.
- **NX_NOT_BOUND** (0x24) This socket is not bound to a port.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer or port return pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Get the port number of previously created and bound client
   socket. */
status = nx_tcp_client_socket_port_get(&client_socket, &port);

/* If status is NX_SUCCESS, the port variable contains the port this
   socket is bound to. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive

- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_client_socket_unbind

Unbind TCP client socket from TCP port

Prototype

```
UINT nx_tcp_client_socket_unbind(NX_TCP_SOCKET *socket_ptr);
```

Description

This service releases the binding between the TCP client socket and a TCP port. If there are other threads waiting to bind another socket to the same port number, the first suspended thread is then bound to this port.

Parameters

- *socket_ptr*: Pointer to previously created TCP socket instance.

Return Values

- **NX_SUCCESS** (0x00) Successful socket unbind.
- **NX_NOT_BOUND** (0x24) Socket was not bound to any port.
- **NX_NOT_CLOSED** (0x35) Socket has not been disconnected.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

Yes

Example

```
/* Unbind a previously created and bound client TCP socket. */
status = nx_tcp_client_socket_unbind(&client_socket);

/* If status is NX_SUCCESS, the client socket is no longer
   bound. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_enable

Enable TCP component of NetX Duo

Prototype

```
UINT nx_tcp_enable(NX_IP *ip_ptr);
```

Description

This service enables the Transmission Control Protocol (TCP) component of NetX Duo. After enabled, TCP connections may be established by the application.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful TCP enable.
- **NX_ALREADY_ENABLED** (0x15) TCP is already enabled.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.

- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Initialization, threads, timers

Preemption Possible

No

Example

```
/* Enable TCP on a previously created IP instance ip_0. */
status = nx_tcp_enable(&ip_0);

/* If status is NX_SUCCESS, TCP is enabled on the IP instance. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_free_port_find

Find next available TCP port

Prototype

```
UINT nx_tcp_free_port_find(
    NX_IP *ip_ptr,
    UINT port,
    UINT *free_port_ptr);
```

Description

This service attempts to locate a free TCP port (unbound) starting from the application supplied port. The search logic will wrap around if the search happens to reach the maximum port value of 0xFFFF. If the search is successful, the free port is returned in the variable pointed to by *free_port_ptr*.

> **Warning:** *This service can be called from another thread and have the same port returned. To prevent this race condition, the application may wish to place this service and the actual client socket bind under the protection of a mutex.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *port*: Port number to start search at (1 through 0xFFFF).
- *free_port_ptr*: Pointer to the destination free port return value.

Return Values

- **NX_SUCCESS** (0x00) Successful free port find.
- **NX_NO_FREE_PORTS** (0x45) No free ports found.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_INVALID_PORT** (0x46) The specified port number is invalid.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Locate a free TCP port, starting at port 12, on a previously
   created IP instance. */
status = nx_tcp_free_port_find(&ip_0, 12, &free_port);
```

```
/* If status is NX_SUCCESS, "free_port" contains the next free port
   on the IP instance. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_info_get

Retrieve information about TCP activities

Prototype

```
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);
```

Description

This service retrieves information about TCP activities for the specified IP instance.

> **Important:** *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *tcp_packets_sent*: Pointer to destination for the total number of TCP packets sent.
- *tcp_bytes_sent*: Pointer to destination for the total number of TCP bytes sent.
- *tcp_packets_received*: Pointer to destination of the total number of TCP packets received.
- *tcp_bytes_received*: Pointer to destination of the total number of TCP bytes received.
- *tcp_invalid_packets*: Pointer to destination of the total number of invalid TCP packets.
- *tcp_receive_packets_dropped*: Pointer to destination of the total number of TCP receive packets dropped.
- *tcp_checksum_errors*: Pointer to destination of the total number of TCP packets with checksum errors.
- *tcp_connections*: Pointer to destination of the total number of TCP connections.
- *tcp_disconnections*: Pointer to destination of the total number of TCP disconnections.
- *tcp_connections_dropped*: Pointer to destination of the total number of TCP connections dropped.
- *tcp_retransmit_packets*: Pointer to destination of the total number of TCP packets retransmitted.

Return Values

- **NX_SUCCESS** (0x00) Successful TCP information retrieval.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Retrieve TCP information from previously created IP Instance
   ip_0. */
status = nx_tcp_info_get(&ip_0,
                         &tcp_packets_sent,
                         &tcp_bytes_sent,
                         &tcp_packets_received,
                         &tcp_bytes_received,
                         &tcp_invalid_packets,
                         &tcp_receive_packets_dropped,
                         &tcp_checksum_errors,
                         &tcp_connections,
                         &tcp_disconnections
                         &tcp_connections_dropped,
                         &tcp_retransmit_packets);

/* If status is NX_SUCCESS, TCP information was retrieved. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_server_socket_accept

Accept TCP connection

Prototype

```
UINT nx_tcp_server_socket_accept(
    NX_TCP_SOCKET *socket_ptr,
    ULONG wait_option);
```

Description

This service accepts (or prepares to accept) a TCP client socket connection request for a port that was previously set up for listening. This service may be called immediately after the application calls the listen or re-listen service or after the listen callback routine is called when the client connection is actually present. If a connection cannot not be established right away, the service suspends according to the supplied wait option.

> **Warning:** *The application must call* **nx_tcp_server_socket_unaccept*** after the connection is no longer needed to remove the server socket's binding to the server port*.
>
> **Important:** *Application callback routines are called from within the IP's helper thread.*

Parameters

- *socket_ptr*: Pointer to the TCP server socket control block.
- *wait_option*: Defines how the service behaves while the connection is being established. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful TCP server socket accept (passive connect).
- **NX_NOT_LISTEN_STATE** (0x36) The server socket supplied is not in a listen state.
- **NX_IN_PROGRESS** (0x37) No wait was specified, the connection attempt is in progress.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_PTR_ERROR** (0x07) Socket pointer error.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
NX_PACKET_POOL my_pool;
NX_IP my_ip;
NX_TCP_SOCKET server_socket;

void port_12_connect_request(NX_TCP_SOCKET *socket_ptr, UINT port)
{
    /* Simply set the semaphore to wake up the server thread. */
    tx_semaphore_put(&port_12_semaphore);
}

void port_12_disconnect_request(NX_TCP_SOCKET *socket_ptr)
{
    /* The client has initiated a disconnect on this socket. This
       example doesn't use this callback. */
}

void port_12_server_thread_entry(ULONG id)
{
NX_PACKET *my_packet;
UINT status, i;
    /* Assuming that:
       "port_12_semaphore" has already been created with an
        initial count of 0 "my_ip" has already been created and the
        link is enabled "my_pool" packet pool has already been
        created
    */

    /* Create the server socket. */
    nx_tcp_socket_create(&my_ip, &server_socket,
                         "Port 12 Server Socket",
                          NX_IP_NORMAL, NX_FRAGMENT_OKAY,
                          NX_IP_TIME_TO_LIVE, 100,
                          NX_NULL, port_12_disconnect_request);

    /* Setup server listening on port 12. */
    nx_tcp_server_socket_listen(&my_ip, 12, &server_socket, 5,
                                port_12_connect_request);
```

```
/* Loop to process 5 server connections, sending
   "Hello_and_Goodbye" to each client and then disconnecting.*/
for (i = 0; i < 5; i++)
{

    /* Get the semaphore that indicates a client connection
    request is present. */
    tx_semaphore_get(&port_12_semaphore, TX_WAIT_FOREVER);

    /* Wait for 200 ticks for the client socket connection to
       complete.*/
    status = nx_tcp_server_socket_accept(&server_socket, 200);

    /* Check for a successful connection. */
    if (status == NX_SUCCESS)
    {

        /* Allocate a packet for the "Hello_and_Goodbye"
           message */
        nx_packet_allocate(&my_pool, &my_packet, NX_TCP_PACKET,
                                    NX_WAIT_FOREVER);

        /* Place "Hello_and_Goodbye" in the packet. */
        nx_packet_data_append(my_packet, "Hello_and_Goodbye",
                              sizeof("Hello_and_Goodbye"),
                              &my_pool, NX_WAIT_FOREVER);

        /* Send "Hello_and_Goodbye" to client. */
        nx_tcp_socket_send(&server_socket, my_packet, 200);

        /* Check for an error. */
        if (status)
        {
           /* Error, release the packet. */
           nx_packet_release(my_packet);
        }

        /* Now disconnect the server socket from the client. */
        nx_tcp_socket_disconnect(&server_socket, 200);
     }

     /* Unaccept the server socket. Note that unaccept is called
        even if disconnect or accept fails. */
     nx_tcp_server_socket_unaccept(&server_socket);
```

```
        /* Setup server socket for listening with this socket
           again. */
        nx_tcp_server_socket_relisten(&my_ip, 12, &server_socket);
    }

    /* We are now done so unlisten on server port 12. */
    nx_tcp_server_socket_unlisten(&my_ip, 12);

    /* Delete the server socket. */
    nx_tcp_socket_delete(&server_socket);
}
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_server_socket_listen

Enable listening for client connection on TCP port

Prototype

```
UINT nx_tcp_server_socket_listen(
    NX_IP *ip_ptr, UINT port,
    NX_TCP_SOCKET *socket_ptr,
```

```
UINT listen_queue_size,
VOID (*listen_callback)(NX_TCP_SOCKET *socket_ptr, UINT port));
```

Description

This service enables listening for a client connection request on the specified TCP port. When a client connection request is received, the supplied server socket is bound to the specified port and the supplied listen callback function is called.

The listen callback routine's processing is completely up to the application. It may contain logic to wake up an application thread that subsequently performs an accept operation. If the application already has a thread suspended on accept processing for this socket, the listen callback routine may not be needed.

If the application wishes to handle additional client connections on the same port, the ***nx_tcp_server_socket_relisten*** must be called with an available socket (a socket in the CLOSED state) for the next connection. Until the re-listen service is called, additional client connections are queued. When the maximum queue depth is exceeded, the oldest connection request is dropped in favor of queuing the new connection request. The maximum queue depth is specified by this service.

> **Important:** *Application callback routines are called from the internal IP helper thread.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *port*: Port number to listen on (1 through 0xFFFF).
- *socket_ptr*: Pointer to socket to use for the connection.
- *listen_queue_size*: Number of client connection requests that can be queued.
- *listen_callback*: Application function to call when the connection is received. If a NULL is specified, the listen callback feature is disabled.

Return Values

- **NX_SUCCESS** (0x00) Successful TCP port listen enable.
- **NX_MAX_LISTEN** (0x33) No more listen request structures are available. The constant NX_MAX_LISTEN_REQUESTS in ***nx_api.h*** defines how many active listen requests are possible.
- **NX_NOT_CLOSED** (0x35) The supplied server socket is not in a closed state.
- **NX_ALREADY_BOUND** (0x22) The supplied server socket is already bound to a port.
- **NX_DUPLICATE_LISTEN** (0x34) There is already an active listen request for this port.

- **NX_INVALID_PORT** (0x46) Invalid port specified.
- **NX_PTR_ERROR** (0x07) Invalid IP or socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
NX_PACKET_POOL my_pool;
NX_IP my_ip;
NX_TCP_SOCKET server_socket;

void port_12_connect_request(NX_TCP_SOCKET *socket_ptr, UINT port)
{
   /* Simply set the semaphore to wake up the server thread.*/
   tx_semaphore_put(&port_12_semaphore);
}

void port_12_disconnect_request(NX_TCP_SOCKET *socket_ptr)
{
   /* The client has initiated a disconnect on this socket.
   This example doesn't use this callback. */
}

void port_12_server_thread_entry(ULONG id)
{
NX_PACKET *my_packet;
UINT status, i;
   /* Assuming that:
      "port_12_semaphore" has already been created with an
      initial count of 0 "my_ip" has already been created
      and the link is enabled "my_pool" packet pool has already
      been created.
   */

   /* Create the server socket. */
   nx_tcp_socket_create(&my_ip, &server_socket, "Port 12 Server
                        Socket",
                        NX_IP_NORMAL, NX_FRAGMENT_OKAY,
                        NX_IP_TIME_TO_LIVE, 100,
```

```
                    NX_NULL, port_12_disconnect_request);

/* Setup server listening on port 12. */
nx_tcp_server_socket_listen(&my_ip, 12, &server_socket, 5,
                            port_12_connect_request);

/* Loop to process 5 server connections, sending
   "Hello_and_Goodbye" to
   each client and then disconnecting. */
for (i = 0; i < 5; i++)
{

     /* Get the semaphore that indicates a client connection
        request is present. */
     tx_semaphore_get(&port_12_semaphore, TX_WAIT_FOREVER);

     /* Wait for 200 ticks for the client socket connection
        to complete. */
     status = nx_tcp_server_socket_accept(&server_socket, 200);

     /* Check for a successful connection. */
        if (status == NX_SUCCESS)
     {

           /* Allocate a packet for the "Hello_and_Goodbye"
              message. */
           nx_packet_allocate(&my_pool, &my_packet, NX_TCP_PACKET,
                              NX_WAIT_FOREVER);

           /* Place "Hello_and_Goodbye" in the packet. */
           nx_packet_data_append(my_packet, "Hello_and_Goodbye",
                                 sizeof("Hello_and_Goodbye"),
                                 &my_pool,
                                 NX_WAIT_FOREVER);

          /* Send "Hello_and_Goodbye" to client. */
          nx_tcp_socket_send(&server_socket, my_packet, 200);

          /* Check for an error. */
          if (status)
          {
             /* Error, release the packet. */
             nx_packet_release(my_packet);
          }

         /* Now disconnect the server socket from the client. */
```

```
            nx_tcp_socket_disconnect(&server_socket, 200);
        }
        /* Unaccept the server socket. Note that unaccept is called
           even if disconnect or accept fails. */
        nx_tcp_server_socket_unaccept(&server_socket);

        /* Setup server socket for listening with this socket
        again. */
        nx_tcp_server_socket_relisten(&my_ip, 12, &server_socket);
    }
    /* We are now done so unlisten on server port 12. */
    nx_tcp_server_socket_unlisten(&my_ip, 12);
    /* Delete the server socket. */
    nx_tcp_socket_delete(&server_socket);
}
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_server_socket_relisten

Re-listen for client connection on TCP port

Prototype

```
UINT nx_tcp_server_socket_relisten(
    NX_IP *ip_ptr,
    UINT port,
    NX_TCP_SOCKET *socket_ptr);
```

Description

This service is called after a connection has been received on a port that was setup previously for listening. The main purpose of this service is to provide a new server socket for the next client connection. If a connection request is queued, the connection will be processed immediately during this service call.

> **Important:** *The same callback routine specified by the original listen request is also called when a connection is present for this new server socket.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *port*: Port number to re-listen on (1 through 0xFFFF).
- *socket_ptr*: Socket to use for the next client connection.

Return Values

- **NX_SUCCESS** (0x00) Successful TCP port re-listen.
- **NX_NOT_CLOSED** (0x35) The supplied server socket is not in a closed state.
- **NX_ALREADY_BOUND** (0x22) The supplied server socket is already bound to a port.
- **NX_INVALID_RELISTEN** (0x47) There is already a valid socket pointer for this port or the port specified does not have a listen request active.
- **NX_CONNECTION_PENDING** (0x48) Same as NX_SUCCESS, except there was a queued connection request and it was processed during this call.
- **NX_INVALID_PORT** (0x46) Invalid port specified.
- **NX_PTR_ERROR** (0x07) Invalid IP or listen callback pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
NX_PACKET_POOL my_pool;
NX_IP my_ip;
NX_TCP_SOCKET server_socket;

void port_12_connect_request(NX_TCP_SOCKET *socket_ptr, UINT port)
{

   /* Simply set the semaphore to wake up the server thread.*/
   tx_semaphore_put(&port_12_semaphore);
}

void port_12_disconnect_request(NX_TCP_SOCKET *socket_ptr)
{
   /* The client has initiated a disconnect on this socket. This
      example doesn't use this callback. */
}

void port_12_server_thread_entry(ULONG id)
{

NX_PACKET *my_packet;
UINT status, i;

   /* Assuming that:
     "port_12_semaphore" has already been created with an initial
     count of 0.
    "my_ip" has already been created and the link is enabled.
    "my_pool" packet pool has already been created. */

   /* Create the server socket. */
   nx_tcp_socket_create(&my_ip, &server_socket, "Port 12 Server Socket",
                                NX_IP_NORMAL, NX_FRAGMENT_OKAY,
                                NX_IP_TIME_TO_LIVE, 100,
                                NX_NULL, port_12_disconnect_request);

   /* Setup server listening on port 12. */
   nx_tcp_server_socket_listen(&my_ip, 12, &server_socket, 5,
                               port_12_connect_request);

   /* Loop to process 5 server connections, sending
```

```
   "Hello_and_Goodbye" to each client then disconnecting. */
for (i = 0; i < 5; i++)
{

    /* Get the semaphore that indicates a client connection
       request is present. */
    tx_semaphore_get(&port_12_semaphore, TX_WAIT_FOREVER);

    /* Wait for 200 ticks for the client socket connection to
       complete. */
    status = nx_tcp_server_socket_accept(&server_socket, 200);

    /* Check for a successful connection. */
       if (status == NX_SUCCESS)
    {

          /* Allocate a packet for the "Hello_and_Goodbye"
             message. */
          nx_packet_allocate(&my_pool, &my_packet, NX_TCP_PACKET,
                             NX_WAIT_FOREVER);

          /* Place "Hello_and_Goodbye" in the packet. */
          nx_packet_data_append(my_packet, "Hello_and_Goodbye",
                                sizeof("Hello_and_Goodbye"),
                                &my_pool, NX_WAIT_FOREVER);

          /* Send "Hello_and_Goodbye" to client. */
          nx_tcp_socket_send(&server_socket, my_packet, 200);

          /* Check for an error. */
          if (status)
          {

             /* Error, release the packet. */
             nx_packet_release(my_packet);
          }

          /* Now disconnect the server socket from the client. */
          nx_tcp_socket_disconnect(&server_socket, 200);
      }

      /* Unaccept the server socket. Note that unaccept is
         called even if disconnect or accept fails. */
      nx_tcp_server_socket_unaccept(&server_socket);

      /* Setup server socket for listening with this socket
```

```
            again. */
        nx_tcp_server_socket_relisten(&my_ip, 12, &server_socket);
    }

    /* We are now done so unlisten on server port 12. */
    nx_tcp_server_socket_unlisten(&my_ip, 12);

    /* Delete the server socket. */
    nx_tcp_socket_delete(&server_socket);
}
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_server_socket_unaccept

Remove socket association with listening port

Prototype

```
UINT nx_tcp_server_socket_unaccept(NX_TCP_SOCKET *socket_ptr);
```

Description

This service removes the association between this server socket and the specified server port. The application must call this service after a disconnection or after an unsuccessful accept call.

Parameters

- *socket_ptr*: Pointer to previously setup server socket instance.

Return Values

- **NX_SUCCESS** (0x00) Successful server socket unaccept.
- **NX_NOT_LISTEN_STATE** (0x36) Server socket is in an improper state, and is probably not disconnected.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
NX_PACKET_POOL        my_pool;
NX_IP                 my_ip;
NX_TCP_SOCKET         server_socket;
void port_12_connect_request(NX_TCP_SOCKET *socket_ptr, UINT port)
{

   /* Simply set the semaphore to wake up the server thread. */
   tx_semaphore_put(&port_12_semaphore);
}

void port_12_disconnect_request(NX_TCP_SOCKET *socket_ptr)
{
   /* The client has initiated a disconnect on this socket. This example
   doesn't use this callback. */
}

void port_12_server_thread_entry(ULONG id)
{
```

```
NX_PACKET  *my_packet;
UINT       status, i;

   /* Assuming that:
     "port_12_semaphore" has already been created with an initial count
      of 0 "my_ip" has already been created and the link is enabled
     "my_pool" packet pool has already been created
   */

    /* Create the server socket. */
    nx_tcp_socket_create(&my_ip, &server_socket, "Port 12 Server
                         Socket",NX_IP_NORMAL, NX_FRAGMENT_OKAY,
                         NX_IP_TIME_TO_LIVE, 100,NX_NULL,
                         port_12_disconnect_request);

    /* Setup server listening on port 12. */
    nx_tcp_server_socket_listen(&my_ip, 12, &server_socket, 5,
                                port_12_connect_request);

    /* Loop to process 5 server connections, sending "Hello_and_Goodbye"
       to
       each client and then disconnecting. */
    for (i = 0; i < 5; i++)
    {

        /* Get the semaphore that indicates a client connection request
           is present. */
        tx_semaphore_get(&port_12_semaphore, TX_WAIT_FOREVER);

        /* Wait for 200 ticks for the client socket connection to
           complete.*/
        status = nx_tcp_server_socket_accept(&server_socket, 200);

       /* Check for a successful connection. */
          if (status == NX_SUCCESS)
       {
             /* Allocate a packet for the "Hello_and_Goodbye" message. */
             nx_packet_allocate(&my_pool, &my_packet, NX_TCP_PACKET,
                                NX_WAIT_FOREVER);

             /* Place "Hello_and_Goodbye" in the packet. */
             nx_packet_data_append(my_packet,
                      "Hello_and_Goodbye",sizeof("Hello_and_Goodbye"),
                      &my_pool, NX_WAIT_FOREVER);

             /* Send "Hello_and_Goodbye" to client. */
```

```
            nx_tcp_socket_send(&server_socket, my_packet, 200);

            /* Check for an error. */
            if (status)
            {

               /* Error, release the packet. */
               nx_packet_release(my_packet);
            }

            /* Now disconnect the server socket from the client. */
            nx_tcp_socket_disconnect(&server_socket, 200);
        }

        /* Unaccept the server socket. Note that unaccept is called even
           if disconnect or accept fails. */
        nx_tcp_server_socket_unaccept(&server_socket);

      /* Setup server socket for listening with this socket again. */
      nx_tcp_server_socket_relisten(&my_ip, 12, &server_socket);
    }

    /* We are now done so unlisten on server port 12. */
    nx_tcp_server_socket_unlisten(&my_ip, 12);

    /* Delete the server socket. */
    nx_tcp_socket_delete(&server_socket);
}
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect

- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_server_socket_unlisten

Disable listening for client connection on TCP port

Prototype

```
UINT nx_tcp_server_socket_unlisten(
    NX_IP *ip_ptr,
    UINT port);
```

Description

This service disables listening for a client connection request on the specified TCP port.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *port*: Number of port to disable listening (0 through 0xFFFF).

Return Values

- **NX_SUCCESS** (0x00) Successful TCP listen disable.
- **NX_ENTRY_NOT_FOUND** (0x16) Listening was not enabled for the specified port.
- **NX_INVALID_PORT** (0x46) Invalid port specified.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
NX_PACKET_POOL       my_pool;
NX_IP                my_ip;
NX_TCP_SOCKET        server_socket;

void port_12_connect_request(NX_TCP_SOCKET *socket_ptr, UINT port)
{

    /* Simply set the semaphore to wake up the server thread. */
    tx_semaphore_put(&port_12_semaphore);
}

void port_12_disconnect_request(NX_TCP_SOCKET *socket_ptr)
{

    /* The client has initiated a disconnect on this socket. This example
       doesn't use this callback.*/
}

void port_12_server_thread_entry(ULONG id)
{

NX_PACKET *my_packet;
UINT status, i;

    /* Assuming that:
      "port_12_semaphore" has already been created with an initial count
       of 0 "my_ip" has already been created and the link is enabled
      "my_pool" packet pool has already been created
    */

    /* Create the server socket. */
    nx_tcp_socket_create(&my_ip, &server_socket, "Port 12 Server Socket",
                         NX_IP_NORMAL, NX_FRAGMENT_OKAY,
                         NX_IP_TIME_TO_LIVE, 100,
                         NX_NULL, port_12_disconnect_request);

    /* Setup server listening on port 12. */
    nx_tcp_server_socket_listen(&my_ip, 12, &server_socket, 5,
                                port_12_connect_request);

    /* Loop to process 5 server connections, sending "Hello_and_Goodbye" to
       each client and then disconnecting. */
    for (i = 0; i < 5; i++)
    {
```

```
    /* Get the semaphore that indicates a client connection request is
       present. */
    tx_semaphore_get(&port_12_semaphore, TX_WAIT_FOREVER);

    /* Wait for 200 ticks for the client socket connection to complete.*/
    status = nx_tcp_server_socket_accept(&server_socket, 200);

    /* Check for a successful connection. */
    if (status == NX_SUCCESS)
    {

        /* Allocate a packet for the "Hello_and_Goodbye" message. */
        nx_packet_allocate(&my_pool, &my_packet, NX_TCP_PACKET,
                           NX_WAIT_FOREVER);

        /* Place "Hello_and_Goodbye" in the packet. */
        nx_packet_data_append(my_packet, "Hello_and_Goodbye",
                              sizeof("Hello_and_Goodbye"), &my_pool,
                              NX_WAIT_FOREVER);

        /* Send "Hello_and_Goodbye" to client. */
        nx_tcp_socket_send(&server_socket, my_packet, 200);

        /* Check for an error. */
        if (status)
        {

           /* Error, release the packet. */
           nx_packet_release(my_packet);
        }

        /* Now disconnect the server socket from the client. */
        nx_tcp_socket_disconnect(&server_socket, 200);
     }

     /* Unaccept the server socket. Note that unaccept is called even if
        disconnect or accept fails. */
     nx_tcp_server_socket_unaccept(&server_socket);

     /* Setup server socket for listening with this socket again. */
     nx_tcp_server_socket_relisten(&my_ip, 12, &server_socket);
}

/* We are now done so unlisten on server port 12. */
nx_tcp_server_socket_unlisten(&my_ip, 12);
```

```
    /* Delete the server socket. */
    nx_tcp_socket_delete(&server_socket);
}
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_bytes_available

Retrieves number of bytes available for retrieval

Prototype

```
UINT nx_tcp_socket_bytes_available(
    NX_TCP_SOCKET *socket_ptr,
    ULONG *bytes_available);
```

Description

This service obtains the number of bytes available for retrieval in the specified TCP socket. Note that the TCP socket must already be connected.

Parameters

- *socket_ptr*: Pointer to previously created and connected TCP socket.
- *bytes_available*: Pointer to destination for bytes available.

Return Values

- **NX_SUCCESS** (0x00) Service executes successfully. Number of bytes available for read is returned to the caller.
- **NX_NOT_CONNECTED** (0x38) Socket is not in a connected state.
- **NX_PTR_ERROR** (0x07) Invalid pointers.
- **NX_NOT_ENABLED** (0x14) TCP is not enabled.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Get the bytes available for retrieval on the specified socket. */
status = nx_tcp_socket_bytes_available(&my_socket,&bytes_available);

/* Is status = NX_SUCCESS, the available bytes is returned in
   bytes_available. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get

- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_create

Create TCP client or server socket

Prototype

```
UINT nx_tcp_socket_create(
    NX_IP *ip_ptr,
    NX_TCP_SOCKET *socket_ptr,
    CHAR *name,
    ULONG type_of_service,
    ULONG fragment,
    UINT time_to_live,
    ULONG window_size,
    VOID (*urgent_data_callback)(NX_TCP_SOCKET *socket_ptr),
    VOID (*disconnect_callback)(NX_TCP_SOCKET *socket_ptr));
```

Description

This service creates a TCP client or server socket for the specified IP instance.

> [!NOTE]
> *Application callback routines are called from the thread associated with this IP instance.*

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *socket_ptr*: Pointer to new TCP socket control block.
- *name*: Application name for this TCP socket.
- *type_of_service*: Defines the type of service for the transmission, legal values are as follows:
  - **NX_IP_NORMAL** (0x00000000)
  - **NX_IP_MIN_DELAY** (0x00100000)
  - **NX_IP_MAX_DATA** (0x00080000)
  - **NX_IP_MAX_RELIABLE** (0x00040000)
  - **NX_IP_MIN_COST** (0x00020000)
- *fragment*: Specifies whether or not IP fragmenting is allowed. **If NX_FRAGMENT_OKAY** (0x0) is specified, IP fragmenting is

allowed. If **NX_DONT_FRAGMENT** (0x4000) is specified, IP fragmenting is disabled.
- *time_to_live*: Specifies the 8-bit value that defines how many routers this packet can pass before being thrown away. The default value is specified by **NX_IP_TIME_TO_LIVE**.
- *window_size** Defines the maximum number of bytes allowed in the receive queue for this socket
- *urgent_data_callback** Application function that is called whenever urgent data is detected in the receive stream. If this value is NX_NULL, urgent data is ignored.
- *disconnect_callback** Application function that is called whenever a disconnect is issued by the socket at the other end of the connection. If this value is NX_NULL, the disconnect callback function is disabled.

Return Values

- **NX_SUCCESS** (0x00) Successful TCP client socket create.
- **NX_OPTION_ERROR** (0x0A) Invalid type-of-service, fragment, invalid window size, or time-tolive option.
- **NX_PTR_ERROR** (0x07) Invalid IP or socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization and Threads

Preemption Possible

No

Example

```
/* Create a TCP client socket on the previously created IP instance,
   with normal delivery, IP fragmentation enabled, 0x80 time to
   live, a 200-byte receive window, no urgent callback routine, and
   the "client_disconnect" routine to handle disconnection initiated
   from the other end of the connection. */
status = nx_tcp_socket_create(&ip_0, &client_socket,
                              "Client Socket",
                              NX_IP_NORMAL, NX_FRAGMENT_OKAY,
                              0x80, 200, NX_NULL
                              client_disconnect);

/* If status is NX_SUCCESS, the client socket is created and ready
   to be bound. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_delete

Delete TCP socket

Prototype

```
UINT nx_tcp_socket_delete(NX_TCP_SOCKET *socket_ptr);
```

Description

This service deletes a previously created TCP socket. If the socket is still bound or connected, the service returns an error code.

Parameters

- *socket_ptr** Previously created TCP socket

Return Values

- **NX_SUCCESS** (0x00) Successful socket delete.
- **NX_NOT_CREATED** (0x27) Socket was not created.
- **NX_STILL_BOUND** (0x42) Socket is still bound.

- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Delete a previously created TCP client socket. */
status = nx_tcp_socket_delete(&client_socket);

/* If status is NX_SUCCESS, the client socket is deleted. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_disconnect

Disconnect client and server socket connections

Prototype

```
UINT nx_tcp_socket_disconnect(
    NX_TCP_SOCKET *socket_ptr,
    ULONG wait_option);
```

Description

This service disconnects an established client or server socket connection. A disconnect of a server socket should be followed by an unaccept request, while a client socket that is disconnected is left in a state ready for another connection request. If the disconnect process cannot finish immediately, the service suspends according to the supplied wait option.

Parameters

- *socket_ptr** Pointer to previously connected client or server socket instance.
- *wait_option** Defines how the service behaves while the disconnection is in progress. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful socket disconnect.
- **NX_NOT_CONNECTED** (0x38) Specified socket is not connected.
- **NX_IN_PROGRESS** (0x37) Disconnect is in progress, no wait was specified.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

Yes

Example

```
/* Disconnect from a previously established connection and wait a
   maximum of 400 timer ticks. */
```

```
status = nx_tcp_socket_disconnect(&client_socket, 400);

/* If status is NX_SUCCESS, the previously connected socket (either
   as a result of the client socket connect or the server accept) is
   disconnected. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_disconnect_complete_notify

Install TCP disconnect complete notify callback function

Prototype

```
UINT nx_tcp_socket_disconnect_complete_notify(
    NX_TCP_SOCKET *socket_ptr,
    VOID (*tcp_disconnect_complete_notify)(NX_TCP_SOCKET *socket_ptr));
```

Description

This service registers a callback function which is invoked after a socket disconnect operation is completed. The TCP socket disconnect complete callback function is available if NetX Duo is built with the option ***NX_ENABLE_EXTENDED_NOTIFY_SUPPORT*** defined.

Parameters

- *socket_ptr** Pointer to previously connected client or server socket instance.
- *tcp_disconnect_complete_notify** The callback function to be installed.

Return Values

- **NX_SUCCESS** (0x00) Successfully registered the callback function.
- **NX_NOT_SUPPORTED** (0x4B) The extended notify feature is not built into the NetX Duo library NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) TCP feature is not enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Install the disconnect complete notify callback function. */
status = nx_tcp_socket_disconnect_complete_notify(&client_socket,
                                                  callback);
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_setnx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_establish_notify

Set TCP establish notify callback function

Prototype

```
UINT nx_tcp_socket_establish_notify(
    NX_TCP_SOCKET *socket_ptr,
    VOID (*tcp_establish_notify)(NX_TCP_SOCKET *socket_ptr));
```

Description

This service registers a callback function, which is called after a TCP socket makes a connection. The TCP socket establish callback function is available if NetX Duo is built with the option ***NX_ENABLE_EXTENDED_NOTIFY_SUPPORT*** defined.

Parameters

- *socket_ptr** Pointer to previously connected client or server socket instance.
- *tcp_establish_notify** Callback function invoked after a TCP connection is established.

Return Values

- **NX_SUCCESS** (0x00) Successfully sets the notify function.
- **NX_NOT_SUPPORTED** (0x4B) The extended notify feature is not built into the NetX Duo library
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) TCP has not been enabled by the application.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Set the function pointer "callback" as the notify function NetX
   Duo will call when the connection is in the established state. */
status = nx_tcp_socket_establish_notify(&client_socket, callback);
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create

- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_info_get

Retrieve information about TCP socket activities

Prototype

```
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);
```

Description

This service retrieves information about TCP socket activities for the specified TCP socket instance.

> [!NOTE]
> *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *socket_ptr** Pointer to previously created TCP socket instance.
- *tcp_packets_sent** Pointer to destination for the total number of TCP packets sent on socket.
- *tcp_bytes_sent** Pointer to destination for the total number of TCP bytes sent on socket.

- *tcp_packets_received** Pointer to destination of the total number of TCP packets received on socket.
- *tcp_bytes_received** Pointer to destination of the total number of TCP bytes received on socket.
- *tcp_retransmit_packets** Pointer to destination of the total number of TCP packet retransmissions.
- *tcp_packets_queued** Pointer to destination of the total number of queued TCP packets on socket.
- *tcp_checksum_errors** Pointer to destination of the total number of TCP packets with checksum errors on socket.
- *tcp_socket_state** Pointer to destination of the socket's current state.
- *tcp_transmit_queue_depth** Pointer to destination of the total number of transmit packets still queued waiting for ACK.
- *tcp_transmit_window** Pointer to destination of the current transmit window size.
- *tcp_receive_window** Pointer to destination of the current receive window size.

Return Values

- **NX_SUCCESS** (0x00) Successful TCP socket information retrieval.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Retrieve TCP socket information from previously created
   socket_0.*/
status = nx_tcp_socket_info_get(&socket_0,
                                &tcp_packets_sent,
                                &tcp_bytes_sent,
                                &tcp_packets_received,
                                &tcp_bytes_received,
                                &tcp_retransmit_packets,
                                &tcp_packets_queued,
                                &tcp_checksum_errors,
                                &tcp_socket_state,
                                &tcp_transmit_queue_depth,
```

```
                                    &tcp_transmit_window,
                                    &tcp_receive_window);

/* If status is NX_SUCCESS, TCP socket information was retrieved. */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_mss_get

Get MSS of socket

Prototype

```
UINT nx_tcp_socket_mss_get(
    NX_TCP_SOCKET *socket_ptr,
    ULONG *mss);
```

Description

This service retrieves the specified socket's local Maximum Segment Size (MSS).

Parameters

- *socket_ptr** Pointer to previously created socket.

- *mss** Destination for returning MSS.

Return Values

- **NX_SUCCESS** (0x00) Successful MSS get.
- **NX_PTR_ERROR** (0x07) Invalid socket or MSS destination pointer.
- **NX_NOT_ENABLED** (0x14) TCP is not enabled.
- **NX_CALLER_ERROR** (0x11) Caller is not a thread or initialization.

Allowed From

Initialization and threads

Preemption Possible

No

Example

```
/* Get the MSS for the socket "my_socket". */
status = nx_tcp_socket_mss_get(&my_socket, &mss_value);

/* If status is NX_SUCCESS, the "mss_value" variable contains the
   socket's current MSS value. */
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_mss_peer_get

Get MSS of the peer TCP socket

Prototype

```
UINT nx_tcp_socket_mss_peer_get(
    NX_TCP_SOCKET *socket_ptr,
```

```
    ULONG *mss);
```

Description

This service retrieves the Maximum Segment Size (MSS) advertised by the peer socket.

Parameters

- *socket_ptr** Pointer to previously created and connected socket.
- *mss** Destination for returning the MSS.

Return Values

- **NX_SUCCESS** (0x00) Successful peer MSS get.
- **NX_PTR_ERROR** (0x07) Invalid socket or MSS destination pointer.
- **NX_NOT_ENABLED** (0x14) TCP is not enabled.
- **NX_CALLER_ERROR** (0x11) Caller is not a thread or initialization.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Get the MSS of the connected peer to the socket "my_socket". */
status = nx_tcp_socket_mss_peer_get(&my_socket, &mss_value);

/* If status is NX_SUCCESS, the "mss_value" variable contains the
   socket peer's advertised MSS value. */
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure

- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_mss_set

Set MSS of socket

Prototype

```
UINT nx_tcp_socket_mss_set(
    NX_TCP_SOCKET *socket_ptr,
    ULONG mss);
```

Description

This service sets the specified socket's Maximum Segment Size (MSS). Note the MSS value must be within the network interface IP MTU, allowing room for IP and TCP headers.

This service should be used before a TCP socket starts the connection process. If the service is used after a TCP connection is established, the new value has no effect on the connection.

Parameters

- *socket_ptr** Pointer to previously created socket.
- *mss** Value of MSS to set.

Return Values

- **NX_SUCCESS** (0x00) Successful MSS set.
- **NX_SIZE_ERROR** (0x09) Specified MSS value is too large.
- **NX_NOT_CONNECTED** (0x38) TCP connection has not been established
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_NOT_ENABLED** (0x14) TCP is not enabled.
- **NX_CALLER_ERROR** (0x11) Caller is not a thread or initialization.

Allowed From

Initialization and threads

Preemption Possible

No

Example

```
/* Set the MSS of the socket "my_socket" to 1000 bytes. */
status = nx_tcp_socket_mss_set(&my_socket, 1000);

/* If status is NX_SUCCESS, the MSS of "my_socket" is 1000 bytes. */
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_peer_info_get

Retrieve information about peer TCP socket

Prototype

```
UINT nx_tcp_socket_peer_info_get(
    NX_TCP_SOCKET *socket_ptr,
    ULONG *peer_ip_address,
    ULONG *peer_port);
```

Description

This service retrieves peer IP address and port information for the connected TCP socket over IPv4 network. The equivalent service that also supports IPv6 network is ***nxd_tcp_socket_peer_info_get.***

Parameters

- *socket_ptr** Pointer to previously created TCP socket.
- *peer_ip_address** Pointer to destination for peer IP address, in host byte order.
- *peer_port** Pointer to destination for peer port number, in host byte order.

Return Values

- **NX_SUCCESS** (0x00) Service executes successfully. Peer IP address and port number are returned to the caller.
- **NX_NOT_CONNECTED** (0x38) Socket is not in a connected state.
- **NX_PTR_ERROR** (0x07) Invalid pointers.
- **NX_NOT_ENABLED** (0x14) TCP is not enabled.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Obtain peer IP address and port on the specified TCP socket. */
status = nx_tcp_socket_peer_info_get(&my_socket, &peer_ip_address,
                                     &peer_port);

/* If status = NX_SUCCESS, the data was successfully obtained. */
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_queue_depth_notify_set

Set the TCP transmit queue notify function

Prototype

```
UINT nx_tcp_socket_queue_depth_notify_set(
              NX_TCP_SOCKET *socket_ptr,
```

```
                VOID(*tcp_socket_queue_depth_notify)(NX_TCP_SOCKET *));
```

Description

This service sets the transmit queue depth update notify function specified by the application, which is called whenever the specified socket determines that it has released packets from the transmit queue such that the queue depth is no longer exceeding its limit. If an application would be blocked on transmit due to queue depth, the callback function serves as a notification to the application that it may start transmitting again. This service is available only if the NetX Duo library is built with the option ***NX_ENABLE_TCP_QUEUE_DEPTH_UPDATE_NOTIFY*** defined.

Parameters

- *socket_ptr** Pointer to the socket structure
- *tcp_socket_queue_depth_notify** The notify function to be installed

Return Values

- **NX_SUCCESS** (0x00) Successfully installed the notify function
- **NX_NOT_SUPPORTED** (0x4B) The TCP socket queue depth notify feature is not built into the NetX Duo library
- **NX_PTR_ERROR** (0x07) Invalid pointer to the socket control block or the notify function
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) TCP feature is not enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
VOID tcp_socket_queue_depth_notify(NX_TCP_SOCKET *socket_ptr)
{
   /* Notify the application to resume sending. */

}
/* Install the TCP transmit queue notify function .*/
status = nxd_tcp_socket_queue_depth_notify_set(&tcp_socket,
                                    tcp_socket_queue_depth_notify);
```

```
/* If status == NX_SUCCESS, the callback function is successfully
   installed. */
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_receive

Receive data from TCP socket

Prototype

```
UINT nx_tcp_socket_receive(
    NX_TCP_SOCKET *socket_ptr,
    NX_PACKET **packet_ptr,
    ULONG wait_option);
```

Description

This service receives TCP data from the specified socket. If no data is queued on the specified socket, the caller suspends based on the supplied wait option.

> **Caution:** *If NX_SUCCESS is returned, the application is responsible for releasing the received packet when it is no longer needed.*

Parameters

- *socket_ptr** Pointer to previously created TCP socket instance.
- *packet_ptr** Pointer to TCP packet pointer.
- *wait_option** Defines how the service behaves if do data are currently queued on this socket. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful socket data receive.
- **NX_NOT_BOUND** (0x24) Socket is not bound yet.
- **NX_NO_PACKET** (0x01) No data received.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_NOT_CONNECTED** (0x38) The socket is no longer connected.
- **NX_PTR_ERROR** (0x07) Invalid socket or return packet pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Receive a packet from the previously created and connected TCP
   client socket. If no packet is available, wait for 200 timer
   ticks before giving up. */
status = nx_tcp_socket_receive(&client_socket, &packet_ptr, 200);

/* If status is NX_SUCCESS, the received packet is pointed to by
   "packet_ptr". */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete

- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_receive_notify

Notify application of received packets

Prototype

```
UINT nx_tcp_socket_receive_notify(
    NX_TCP_SOCKET *socket_ptr,
    VOID (*tcp_receive_notify)(NX_TCP_SOCKET *socket_ptr));
```

Description

This service configures the receive notify function pointer with the callback function specified by the application. This callback function is then called whenever one or more packets are received on the socket. If a NX_NULL pointer is supplied, the notify function is disabled.

Parameters

- *socket_ptr** Pointer to the TCP socket.
- *tcp_receive_notify** Application callback function pointer that is called when one or more packets are received on the socket.

Return Values

- **NX_SUCCESS** (0x00) Successful socket receive notify.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) TCP feature is not enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Setup a receive packet callback function for the "client_socket"
   socket. */
status = nx_tcp_socket_receive_notify(&client_socket,
                                      my_receive_notify);

/* If status is NX_SUCCESS, NetX Duo will call the function named
   "my_receive_notify" whenever one or more packets are received for
   "client_socket". */
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_send

Send data through a TCP socket

Prototype

```
UINT nx_tcp_socket_send(
    NX_TCP_SOCKET *socket_ptr,
    NX_PACKET *packet_ptr,
    ULONG wait_option);
```

Description

This service sends TCP data through a previously connected TCP socket. If the receiver's last advertised window size is less than this request, the service optionally suspends based on the wait option specified. This service guarantees that no packet data larger than MSS is sent to the IP layer.

> **Warning:** *Unless an error is returned, the application should not release the packet after this call. Doing so will cause unpredictable results because the network driver will also try to release the packet after transmission.*

Parameters

- *socket_ptr** Pointer to previously connected TCP socket instance.
- *packet_ptr** TCP data packet pointer.
- *wait_option** Defines how the service behaves if the request is greater than the window size of the receiver. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful socket send.
- **NX_NOT_BOUND** (0x24) Socket was not bound to any port.
- **NX_NO_INTERFACE_ADDRESS** (0x50) No suitable outgoing interface found.
- **NX_NOT_CONNECTED** (0x38) Socket is no longer connected.
- **NX_WINDOW_OVERFLOW** (0x39) Request is greater than receiver's advertised window size in bytes.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_INVALID_PACKET** (0x12) Packet is not allocated.
- **NX_TX_QUEUE_DEPTH** (0x49) Maximum transmit queue depth has been reached.
- **NX_OVERFLOW** (0x03) Packet append pointer is invalid.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_UNDERFLOW** (0x02) Packet prepend pointer is invalid.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Send a packet out on the previously created and connected TCP
   socket. If the receive window on the other side of the connection
   is less than the packet size, wait 200 timer ticks before giving
   up. */
status = nx_tcp_socket_send(&client_socket, packet_ptr, 200);
```

```
/* If status is NX_SUCCESS, the packet has been sent! */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_state_wait

Wait for TCP socket to enter specific state

Prototype

```
UINT nx_tcp_socket_state_wait(
    NX_TCP_SOCKET *socket_ptr,
    UINT desired_state,
    ULONG wait_option);
```

Description

This service waits for the socket to enter the desired state. If the socket is not in the desired state, the service suspends according to the supplied wait option.

Parameters

- *socket_ptr** Pointer to previously connected TCP socket instance.

- *desired_state** Desired TCP state. Valid TCP socket states are defined as follows:
  - **NX_TCP_CLOSED** (0x01)
  - **NX_TCP_LISTEN_STATE** (0x02)
  - **NX_TCP_SYN_SENT** (0x03)
  - **NX_TCP_SYN_RECEIVED** (0x04)
  - **NX_TCP_ESTABLISHED** (0x05)
  - **NX_TCP_CLOSE_WAIT** (0x06)
  - **NX_TCP_FIN_WAIT_1** (0x07)
  - **NX_TCP_FIN_WAIT_2** (0x08)
  - **NX_TCP_CLOSING** (0x09)
  - **NX_TCP_TIMED_WAIT** (0x0A)
  - **NX_TCP_LAST_ACK** (0x0B)
- *wait_option** Defines how the service behaves if the requested state is not present. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFF)

Return Values

- **NX_SUCCESS** (0x00) Successful state wait.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_NOT_SUCCESSFUL** (0x43) State not present within the specified wait time.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_OPTION_ERROR** (0x0A) The desired socket state is invalid.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Wait 300 timer ticks for the previously created socket to enter
   the established state in the TCP state machine. */
status = nx_tcp_socket_state_wait(&client_socket,
                                  NX_TCP_ESTABLISHED, 300);

/* If status is NX_SUCCESS, the socket is now in the established
   state! */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nxd_tcp_client_socket_connect
- nxd_tcp_socket_peer_info_get

nx_tcp_socket_timed_wait_callback

Install callback for timed wait state

Prototype

```
UINT nx_tcp_socket_timed_wait_callback(
    NX_TCP_SOCKET *socket_ptr,
    VOID (*tcp_timed_wait_callback)(NX_TCP_SOCKET *socket_ptr));
```

Description

This service registers a callback function which is invoked when the TCP socket is in timed wait state. To use this service, the NetX Duo library must be built with the option ***NX_ENABLE_EXTENDED_NOTIFY*** defined.

Parameters

- *socket_ptr** Pointer to previously connected client or server socket instance.
- *tcp_timed_wait_callback** The timed wait callback function

Return Values

- **NX_SUCCESS** (0x00) Successfully registers the callback function socket
- **NX_NOT_SUPPORTED** (0x4B) NetX Duo library is built without the extended notify feature enabled.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) TCP feature is not enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Install the timed wait callback function */
nx_tcp_socket_timed_wait_callback(&client_socket, callback);
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_receive_notify
- nx_tcp_socket_transmit_configure
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_transmit_configure

Configure socket's transmit parameters

Prototype

```
UINT nx_tcp_socket_transmit_configure(
    NX_TCP_SOCKET *socket_ptr,
    ULONG max_queue_depth,
    ULONG timeout,
```

```
    ULONG max_retries,
    ULONG timeout_shift);
```

Description

This service configures various transmit parameters of the specified TCP socket.

Parameters

- *socket_ptr** Pointer to the TCP socket.
- *max_queue_depth** Maximum number of packets allowed to be queued for transmission.
- *timeout** Number of ThreadX timer ticks an ACK is waited for before the packet is sent again.
- *max_retries** Maximum number of retries allowed.
- *timeout_shift** Value to shift the timeout for each subsequent retry. A value of 0, results in the same timeout between successive retries. A value of 1, doubles the timeout between retries.

Return Values

- **NX_SUCCESS** (0x00) Successful transmit socket configure.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_OPTION_ERROR** (0x0a) Invalid queue depth option.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) TCP feature is not enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Configure the "client_socket" for a maximum transmit queue depth of
   12, 100 tick timeouts, a maximum of 20 retries, and a timeout double
   on each successive retry. */
status = nx_tcp_socket_transmit_configure(&client_socket,12,100,20,1);

/* If status is NX_SUCCESS, the socket's transmit retry has been
   configured. */
```

See Also

- nx_tcp_enable

- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_window_update_notify_set

nx_tcp_socket_window_update_notify_set

Notify application of window size updates

Prototype

```
UINT nx_tcp_socket_window_update_notify_set(
    NX_TCP_SOCKET *socket_ptr,
    VOID (*tcp_window_update_notify)(NX_TCP_SOCKET *socket_ptr));
```

Description

This service installs a socket window update callback routine. This routine is called automatically whenever the specified socket receives a packet indicating an increase in the window size of the remote host.

Parameters

- *socket_ptr** Pointer to previously created TCP socket.
- *tcp_window_update_notify** Callback routine to be called when the window size changes. A value of NULL disables the window change update.

Return Values

- **NX_SUCCESS** (0x00) Callback routine is installed on the socket.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_PTR_ERROR** (0x07) Invalid pointers.
- **NX_NOT_ENABLED** (0x14) TCP feature is not enabled.

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Set the function pointer to the windows update callback after creating the
   socket. */
status = nx_tcp_socket_window_update_notify_set(&data_socket,
                                                my_windows_update_callback);

/* Define the window callback function in the host application. */
void my_windows_update_callback(&data_socket)
{

/* Process update on increase TCP transmit socket window size. */
   return;
}
```

See Also

- nx_tcp_enable
- nx_tcp_socket_create
- nx_tcp_socket_disconnect_complete_notify
- nx_tcp_socket_establish_notify
- nx_tcp_socket_mss_get
- nx_tcp_socket_mss_peer_get
- nx_tcp_socket_mss_set
- nx_tcp_socket_peer_info_get
- nx_tcp_socket_queue_depth_notify_set
- nx_tcp_socket_receive_notify
- nx_tcp_socket_timed_wait_callback
- nx_tcp_socket_transmit_configure

nx_udp_enable

Enable UDP component of NetX Duo

Prototype

```
UINT nx_udp_enable(NX_IP *ip_ptr);
```

Description

This service enables the User Datagram Protocol (UDP) component of NetX Duo. After enabled, UDP datagrams may be sent and received by the application.

Parameters

- *ip_ptr** Pointer to previously created IP instance.

Return Values

- **NX_SUCCESS** (0x00) Successful UDP enable.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_ALREADY_ENABLED** (0x15) This component has already been enabled.

Allowed From

Initialization, threads, timers

Preemption Possible

No

Example

```
/* Enable UDP on the previously created IP instance. */
status = nx_udp_enable(&ip_0);

/* If status is NX_SUCCESS, UDP is now enabled on the specified IP
   instance. */
```

See Also

- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_free_port_find

Find next available UDP port

Prototype

```
UINT nx_udp_free_port_find(
    NX_IP *ip_ptr,
    UINT port,
    UINT *free_port_ptr);
```

Description

This service looks for a free UDP port (unbound) starting from the application supplied port number. The search logic will wrap around if the search reaches the maximum port value of 0xFFFF. If the search is successful, the free port is returned in the variable pointed to by free_port_ptr.

> **Warning:** *This service can be called from another thread and can have the same port returned. To prevent this race condition, the application may wish to place this service and the actual socket bind under the protection of a mutex.*

Parameters

- *ip_ptr** Pointer to previously created IP instance.
- *port** Port number to start search (1 through 0xFFFF).
- *free_port_ptr** Pointer to the destination free port return variable.

Return Values

- **NX_SUCCESS** (0x00) Successful free port find.
- **NX_NO_FREE_PORTS** (0x45) No free ports found.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.
- **NX_INVALID_PORT** (0x46) Specified port number is invalid.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Locate a free UDP port, starting at port 12, on a previously
   created IP instance. */
status = nx_udp_free_port_find(&ip_0, 12, &free_port);

/* If status is NX_SUCCESS pointer, "free_port" identifies the next
   free UDP port on the IP instance. */
```

See Also

- nx_udp_enable
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_info_get

Retrieve information about UDP activities

Prototype

```
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);
```

Description

This service retrieves information about UDP activities for the specified IP instance.

> **Important:** *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *ip_ptr** Pointer to previously created IP instance.
- *udp_packets_sent** Pointer to destination for the total number of UDP packets sent.
- *udp_bytes_sent** Pointer to destination for the total number of UDP bytes sent.
- *udp_packets_received** Pointer to destination of the total number of UDP packets received.
- *udp_bytes_received** Pointer to destination of the total number of UDP bytes received.
- *udp_invalid_packets** Pointer to destination of the total number of invalid UDP packets.
- *udp_receive_packets_dropped** Pointer to destination of the total number of UDP receive packets dropped.
- *udp_checksum_errors** Pointer to destination of the total number of UDP packets with checksum errors.

Return Values

- **NX_SUCCESS** (0x00) Successful UDP information retrieval.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads, and timers

Preemption Possible

No

Example

```
/* Retrieve UDP information from previously created IP Instance
   ip_0. */
```

```
status = nx_udp_info_get(&ip_0, &udp_packets_sent,
                         &udp_bytes_sent,
                         &udp_packets_received,
                         &udp_bytes_received,
                         &udp_invalid_packets,
                         &udp_receive_packets_dropped,
                         &udp_checksum_errors);

/* If status is NX_SUCCESS, UDP information was retrieved. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_packet_info_extract

Extract network parameters from UDP packet

Prototype

```
UINT nx_udp_packet_info_extract(
    NX_PACKET *packet_ptr,
    ULONG *ip_address,
    UINT *protocol,
    UINT *port,
    UINT *interface_index);
```

Description

This service extracts network parameters, such as IPv4 address, peer port number, protocol type (this service always returns UDP type) from a packet received on an incoming interface. To obtain information on a packet coming from IPv4 or IPv6 network, application shall use the service ***nxd_udp_packet_info_extract.***

Parameters

- *packet_ptr** Pointer to packet.
- *ip_address** Pointer to sender IP address.
- *protocol** Pointer to protocol (UDP).
- *port** Pointer to sender's port number.
- *interface_index** Pointer to receiving interface index.

Return Values

- **NX_SUCCESS** (0x00) Packet interface data successfully extracted.
- **NX_INVALID_PACKET** (0x12) Packet does not contain IPv4 frame.
- **NX_PTR_ERROR** (0x07) Invalid pointer input
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Extract network data from UDP packet interface.*/
status = nx_udp_packet_info_extract(packet_ptr, &ip_address,
                                    &protocol, &port,
                                    &interface_index);

/* If status is NX_SUCCESS packet data was successfully
   retrieved. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable

- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_bind

Bind UDP socket to UDP port

Prototype

```
UINT nx_udp_socket_bind(
    NX_UDP_SOCKET *socket_ptr,
    UINT port,
    ULONG wait_option);
```

Description

This service binds the previously created UDP socket to the specified UDP port. Valid UDP sockets range from 0 through 0xFFFF. If the requested port number is bound to another socket, this service waits for specified period of time for the socket to unbind from the port number.

Parameters

- *socket_ptr** Pointer to previously created UDP socket instance.
- *port **Port number to bind to** (1 through 0xFFFF). If port number is NX_ANY_PORT** (0x0000), the IP instance will search for the next free port and use that for the binding.
- *wait_option** Defines how the service behaves if the port is already bound to another socket. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful socket bind.
- **NX_ALREADY_BOUND** (0x22) This socket is already bound to another port.
- **NX_PORT_UNAVAILABLE** (0x23) Port is already bound to a different socket.
- **NX_NO_FREE_PORTS** (0x45) No free port.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_INVALID_PORT** (0x46) Invalid port specified.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Bind the previously created UDP socket to port 12 on the
   previously created IP instance. If the port is already bound,
   wait for 300 timer ticks before giving up. */
status = nx_udp_socket_bind(&udp_socket, 12, 300);

/* If status is NX_SUCCESS, the UDP socket is now bound to
   port 12.*/
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive

- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_bytes_available

Retrieves number of bytes available for retrieval

Prototype

```
UINT nx_udp_socket_bytes_available(
    NX_UDP_SOCKET *socket_ptr,
    ULONG *bytes_available);
```

Description

This service retrieves number of bytes available for reception in the specified UDP socket.

Parameters

- *socket_ptr** Pointer to previously created UDP socket.
- *bytes_available** Pointer to destination for bytes available.

Return Values

- **NX_SUCCESS** (0x00) Successful bytes available retrieval.
- **NX_NOT_SUCCESSFUL** (0x43) Socket not bound to a port.
- **NX_PTR_ERROR** (0x07) Invalid pointers.
- **NX_NOT_ENABLED** (0x14) UDP feature is not enabled.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Get the bytes available for retrieval from the UDP socket. */
status = nx_udp_socket_bytes_available(&my_socket,
                                       &bytes_available);

/* If status == NX_SUCCESS, the number of bytes was successfully
   retrieved.*/
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_checksum_disable

Disable checksum for UDP socket

Prototype

```
UINT nx_udp_socket_checksum_disable(NX_UDP_SOCKET *socket_ptr);
```

Description

This service disables the checksum logic for sending and receiving packets on the specified UDP socket. When the checksum logic is disabled, a value of zero is loaded into the UDP header's checksum field for all packets sent through this

socket. A zero-value checksum value in the UDP header signals the receiver that checksum is not computed for this packet.

Also note that this has no effect if **NX_DISABLE_UDP_RX_CHECKSUM** and **NX_DISABLE_UDP_TX_CHECKSUM** are defined when receiving and sending UDP packets respectively,

Note that this service has no effect on packets on the IPv6 network since UDP checksum is mandatory for IPv6.

Parameters

- *socket_ptr** Pointer to previously created UDP socket instance.

Return Values

- **NX_SUCCESS** (0x00) Successful socket checksum disable.
- **NX_NOT_BOUND** (0x24) Socket is not bound.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads, timer

Preemption Possible

No

Example

```
/* Disable the UDP checksum logic for packets sent on this socket. */
status = nx_udp_socket_checksum_disable(&udp_socket);

/* If status is NX_SUCCESS, outgoing packets will not have a checksum
   calculated. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_enable
- nx_udp_socket_create

- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_checksum_enable

Enable checksum for UDP socket

Prototype

```
UINT nx_udp_socket_checksum_enable(NX_UDP_SOCKET *socket_ptr);
```

Description

This service enables the checksum logic for sending and receiving packets on the specified UDP socket. The checksum covers the entire UDP data area as well as a pseudo IP header.

Also note that this has no effect if **NX_DISABLE_UDP_RX_CHECKSUM** and **NX_DISABLE_UDP_TX_CHECKSUM** are defined when receiving and sending UDP packets respectively.

Note that this service has no effect on packets on the IPv6 network. UDP checksum is mandatory in IPv6.

Parameters

- *socket_ptr** Pointer to previously created UDP socket instance.

Return Values

- **NX_SUCCESS** (0x00) Successful socket checksum enable.
- **NX_NOT_BOUND** (0x24) Socket is not bound.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads, timer

Preemption Possible

No

Example

```
/* Enable the UDP checksum logic for packets sent on this socket. */
status = nx_udp_socket_checksum_enable(&udp_socket);

/* If status is NX_SUCCESS, outgoing packets will have a checksum
   calculated. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_create

Create UDP socket

Prototype

```
UINT nx_udp_socket_create(
    NX_IP *ip_ptr,
```

```
NX_UDP_SOCKET *socket_ptr,
CHAR *name,
ULONG type_of_service,
ULONG fragment,
UINT time_to_live,
ULONG queue_maximum);
```

Description

This service creates a UDP socket for the specified IP instance.

Parameters

- *ip_ptr** Pointer to previously created IP instance.
- *socket_ptr** Pointer to new UDP socket control bloc.
- *name** Application name for this UDP socket.
- *type_of_service** Defines the type of service for the transmission, legal values are as follows:
  - **NX_IP_NORMAL** (0x00000000)
  - **NX_IP_MIN_DELAY** (0x00100000)
  - **NX_IP_MAX_DATA** (0x00080000)
  - **NX_IP_MAX_RELIABLE** (0x00040000)
  - **NX_IP_MIN_COST** (0x00020000)
- *fragment Specifies*: whether or not IP fragmenting is allowed. If NX_FRAGMENT_OKAY (0x0) is specified, IP fragmenting is allowed. If NX_DONT_FRAGMENT (0x4000) is specified, IP fragmenting is disabled.
- *time_to_live*: Specifies the 8-bit value that defines how many routers this packet can pass before being thrown away. The default value is specified by **NX_IP_TIME_TO_LIVE**.
- *queue_maximum*: Defines the maximum number of UDP datagrams that can be queued for this socket. After the queue limit is reached, for every new packet received the oldest UDP packet is released.

Return Values

- **NX_SUCCESS** (0x00) Successful UDP socket create.
- **NX_OPTION_ERROR** (0x0A) Invalid type-of-service, fragment, or time-to-live option.
- **NX_PTR_ERROR** (0x07) Invalid IP or socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization and Threads

Preemption Possible

No

Example

```
/* Create a UDP socket with a maximum receive queue of 30 packets.*/
status = nx_udp_socket_create(&ip_0, &udp_socket, "Sample UDP Socket",
                              NX_IP_NORMAL, NX_FRAGMENT_OKAY, 0x80,
                              30);

/* If status is NX_SUCCESS, the new UDP socket has been created and
   is ready for binding. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_delete

Delete UDP socket

Prototype

```
UINT nx_udp_socket_delete(NX_UDP_SOCKET *socket_ptr);
```

Description

This service deletes a previously created UDP socket. If the socket was bound to a port, the socket must be unbound first.

Parameters

- *socket_ptr*: Pointer to previously created UDP socket instance.

Return Values

- **NX_SUCCESS** (0x00) Successful socket delete.
- **NX_STILL_BOUND** (0x42) Socket is still bound.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Delete a previously created UDP socket. */
status = nx_udp_socket_delete(&udp_socket);

/* If status is NX_SUCCESS, the previously created UDP socket has
   been deleted. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify

- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_info_get

Retrieve information about UDP socket activities

Prototype

```
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);
```

Description

This service retrieves information about UDP socket activities for the specified UDP socket instance.

> **Important:** *If a destination pointer is NX_NULL, that particular information is not returned to the caller.*

Parameters

- *socket_ptr*: Pointer to previously created UDP socket instance.
- *udp_packets_sent*: Pointer to destination for the total number of UDP packets sent on socket.
- *udp_bytes_sent*: Pointer to destination for the total number of UDP bytes sent on socket.
- *udp_packets_received*: Pointer to destination of the total number of UDP packets received on socket.
- *udp_bytes_received*: Pointer to destination of the total number of UDP bytes received on socket.
- *udp_packets_queued*: Pointer to destination of the total number of queued UDP packets on socket.

- *udp_receive_packets_dropped*: Pointer to destination of the total number of UDP receive packets dropped for socket due to queue size being exceeded.
- *udp_checksum_errors*: Pointer to destination of the total number of UDP packets with checksum errors on socket.

Return Values

- **NX_SUCCESS** (0x00) Successful UDP socket information retrieval.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Initialization, threads, and timers

Preemption Possible

No

Example

```
/* Retrieve UDP socket information from socket 0.*/
status = nx_udp_socket_info_get(&socket_0,
                                &udp_packets_sent,
                                &udp_bytes_sent,
                                &udp_packets_received,
                                &udp_bytes_received,
                                &udp_queued_packets,
                                &udp_receive_packets_dropped,
                                &udp_checksum_errors);

/* If status is NX_SUCCESS, UDP socket information was retrieved.*/
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_port_get

- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_port_get

Pick up port number bound to UDP socket

Prototype

```
UINT nx_udp_socket_port_get(
    NX_UDP_SOCKET *socket_ptr,
    UINT *port_ptr);
```

Description

This service retrieves the port number associated with the socket, which is useful to find the port allocated by NetX Duo in situations where the NX_ANY_PORT was specified at the time the socket was bound.

Parameters

- *socket_ptr*: Pointer to previously created UDP socket instance.
- *port_ptr*: Pointer to destination for the return port number. Valid port numbers are (1- 0xFFFF).

Return Values

- **NX_SUCCESS** (0x00) Successful socket bind.
- **NX_NOT_BOUND** (0x24) This socket is not bound to a port.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer or port return pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Get the port number of created and bound UDP socket. */
status = nx_udp_socket_port_get(&udp_socket, &port);

/* If status is NX_SUCCESS, the port variable contains the port this
   socket is bound to. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_receive

Receive datagram from UDP socket

Prototype

```
UINT nx_udp_socket_receive(
    NX_UDP_SOCKET *socket_ptr,
    NX_PACKET **packet_ptr,
    ULONG wait_option);
```

Description

This service receives an UDP datagram from the specified socket. If no datagram is queued on the specified socket, the caller suspends based on the supplied wait

option.

> **Caution:** *If NX_SUCCESS is returned, the application is responsible for releasing the received packet when it is no longer needed.*

Parameters

- *socket_ptr*: Pointer to previously created UDP socket instance.
- *packet_ptr*: Pointer to UDP datagram packet pointer.
- *wait_option*: Defines how the service behaves if a datagram is not currently queued on this socket. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful socket receive.
- **NX_NOT_BOUND** (0x24) Socket was not bound to any port.
- **NX_NO_PACKET** (0x01) There was no UDP datagram to receive.
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort.
- **NX_PTR_ERROR** (0x07) Invalid socket or packet return pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

No

Example

```
/* Receive a packet from a previously created and bound UDP socket.
   If no packets are currently available, wait for 500 timer ticks
   before giving up. */
status = nx_udp_socket_receive(&udp_socket, &packet_ptr, 500);

/* If status is NX_SUCCESS, the received UDP packet is pointed to by
   packet_ptr. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find

- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_receive_notify

Notify application of each received packet

Prototype

```
UINT nx_udp_socket_receive_notify(
    NX_UDP_SOCKET *socket_ptr,
    VOID (*udp_receive_notify)(NX_UDP_SOCKET *socket_ptr));
```

Description

This service sets the receive notify function pointer to the callback function specified by the application. This callback function is then called whenever a packet is received on the socket. If a NX_NULL pointer is supplied, the receive notify function is disabled.

Parameters

- *socket_ptr*: Pointer to the UDP socket.
- *udp_receive_notify*: Application callback function pointer that is called when a packet is received on the socket.

Return Values

- **NX_SUCCESS** (0x00) Successfully set socket receive notify function.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.

Allowed From

Initialization, threads, timers, and ISRs

Preemption Possible

No

Example

```
/* Setup a receive packet callback function for the "udp_socket"
   socket. */
status = nx_udp_socket_receive_notify(&udp_socket,
                                      my_receive_notify);

/* If status is NX_SUCCESS, NetX Duo will call the function named
   "my_receive_notify" whenever a packet is received for
   "udp_socket". */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_send

Send a UDP Datagram

Prototype

```
UINT nx_udp_socket_send(
    NX_UDP_SOCKET *socket_ptr,
    NX_PACKET *packet_ptr,
    ULONG ip_address,
    UINT port);
```

Description

This service sends a UDP datagram through a previously created and bound UDP socket for IPv4 networks. NetX Duo finds a suitable local IP address as source address based on the destination IP address. To specify a specific interface and source IP address, the application should use the **nxd_udp_socket_source_send** service.

Note that this service returns immediately regardless of whether the UDP datagram was successfully sent. The NetX Duo (IPv4/IPv6) equivalent service is ***nxd_udp_socket_send***.

The socket must be bound to a local port.

> **Warning:** *Unless an error is returned, the application should not release the packet after this call. Doing so will cause unpredictable results because the network driver will also try to release the packet after transmission.*

Parameters

- *socket_ptr*: Pointer to previously created UDP socket instance
- *packet_ptr*: UDP datagram packet pointer
- *ip_address*: Destination IPv4 address
- *port*: Valid destination port number between 1 and 0xFFFF), in host byte order

Return Values

- **NX_SUCCESS** (0x00) Successful UDP socket send
- **NX_NOT_BOUND** (0x24) Socket not bound to any port
- **NX_NO_INTERFACE_ADDRESS** (0x50) No suitable outgoing interface can be found.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid server IP address
- **NX_UNDERFLOW** (0x02) Not enough room for UDP header in the packet
- **NX_OVERFLOW** (0x03) Packet append pointer is invalid
- **NX_PTR_ERROR** (0x07) Invalid socket pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_NOT_ENABLED** (0x14) UDP has not been enabled

- **NX_INVALID_PORT** (0x46) Port number is not within a valid range

Allowed From

Threads

Preemption Possible

No

Example

```
ULONG server_address;

/* Set the UDP Client IP address. */
server_address = IP_ADDRESS(1,2,3,5);

/* Send a packet to the UDP server at server_address on port 12. */
status = nx_udp_socket_send(&client_socket, packet_ptr,
                            server_address, 12);

/* If status == NX_SUCCESS, the application successfully transmitted
   the packet out the UDP socket to its peer. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_source_send

Send datagram through UDP socket

Prototype

```
UINT nx_udp_socket_source_send(
    NX_UDP_SOCKET *socket_ptr,
    NX_PACKET *packet_ptr,
    ULONG ip_address,
    UINT port,
    UINT address_index);
```

Description

This service sends a UDP datagram through a previously created and bound UDP socket through the network interface with the specified IP address as the source address. Note that service returns immediately, regardless of whether or not the UDP datagram was successfully sent. ***nxd_udp_socket_source_send*** works for both IPv4 and IPv6 networks.

> **Warning:** *Unless an error is returned, the application should not release the packet after this call. Doing so will cause unpredictable results because the network driver will also try to release the packet after transmission.*

Parameters

- *socket_ptr*: Socket to transmit the packet out on.
- *packet_ptr*: Pointer to packet to transmit.
- *ip_address*: Destination IP address to send packet.
- *port*: Destination port.
- *address_index*: Index of the address associated with the interface to send packet on.

Return Values

- **NX_SUCCESS** (0x00) Packet successfully sent.
- **NX_NOT_BOUND** (0x24) Socket not bound to a port.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IP address.
- **NX_NOT_ENABLED** (0x14) UDP processing not enabled.
- **NX_PTR_ERROR** (0x07) Invalid pointer.
- **NX_OVERFLOW** (0x03) Invalid packet append pointer.
- **NX_UNDERFLOW** (0x02) Invalid packet prepend pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_INVALID_INTERFACE** (0x4C) Invalid address index.
- **NX_INVALID_PORT** (0x46) Port number exceeds maximum port number.

Allowed From

Threads

Preemption Possible

No

Example

```
#define ADDRESS_INDEX 1

/* Send packet out on port 80 to the specified destination IP on the
   interface at index 1 in the IP task interface list. */
status = nx_udp_socket_source_send(socket_ptr, packet_ptr,
                                   destination_ip, 80,
                                   ADDRESS_INDEX);

/* If status is NX_SUCCESS packet was successfully transmitted. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_socket_unbind

Unbind UDP socket from UDP port

Prototype

```
UINT nx_udp_socket_unbind(NX_UDP_SOCKET *socket_ptr);
```

Description

This service releases the binding between the UDP socket and a UDP port. Any received packets stored in the receive queue are released as part of the unbind operation.

If there are other threads waiting to bind another socket to the unbound port, the first suspended thread is then bound to the newly unbound port.

Parameters

- *socket_ptr*: Pointer to previously created UDP socket instance.

Return Values

- **NX_SUCCESS** (0x00) Successful socket unbind.
- **NX_NOT_BOUND** (0x24) Socket was not bound to any port.
- **NX_PTR_ERROR** (0x07) Invalid socket pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service.
- **NX_NOT_ENABLED** (0x14) This component has not been enabled.

Allowed From

Threads

Preemption Possible

Yes

Example

```
/* Unbind the previously bound UDP socket. */
status = nx_udp_socket_unbind(&udp_socket);

/* If status is NX_SUCCESS, the previously bound socket is now
   unbound. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available

- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nx_udp_source_extract

Extract IP and sending port from UDP datagram

Prototype

```
UINT nx_udp_source_extract(
    NX_PACKET *packet_ptr,
    ULONG *ip_address,
    UINT *port);
```

Description

This service extracts the sender's IP and port number from the IP and UDP headers of the supplied UDP datagram. Note that the service ***nxd_udp_source_extract*** works with packets from either IPv4 or IPv6 network.

Parameters

- *packet_ptr*: UDP datagram packet pointer.
- *ip_address*: Valid pointer to the return IP address variable.
- *port*: Valid pointer to the return port variable.

Return Values

- **NX_SUCCESS** (0x00) Successful source IP/port extraction.
- **NX_INVALID_PACKET** (0x12) The supplied packet is invalid.
- **NX_PTR_ERROR** (0x07) Invalid packet or IP or port destination.

Allowed From

Initialization, threads, timers, ISR

Preemption Possible

No

Example

```
/* Extract the IP and port information from the sender of the UPD packet. */
status = nx_udp_source_extract(packet_ptr, &sender_ip_address, &sender_port);

/* If status is NX_SUCCESS, the sender IP and port information has been stored
   in sender_ip_address and sender_port respectively.*/
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nxd_icmp_enable

Enable ICMPv4 and ICMPv6 Services

Prototype

```
UINT nxd_icmp_enable(NX_IP *ip_ptr);
```

Description

This service enables both ICMPv4 and ICMPv6 services and can only be called after the IP instance has been created. The service can be enabled either before or after IPv6 is enabled (see *nxd_ipv6_enable*). ICMPv4 services include Echo Request/Reply. ICMPv6 services include Echo Request/Reply, Neighbor Discovery, Duplicate Address Detection, Router Discovery, and Stateless Address Auto-configuration. The IPv4 equivalent in NetX is *nx_icmp_enable*.

> [!NOTE] *If the IPv6 address is manually configured prior to enabling ICMPv6, the manually configured IPv6 is not subject to Duplicate Address Detection process.*

nx_icmp_enable starts ICMP services for IPv4 operations only. Applications using ICMPv6 services must use *nxd_icmp_enable* instead of *nx_icmp_enable*.

To utilize IPv6 router solicitation and IPv6 stateless auto-address configuration, ICMPv6 must be enabled.

Parameters

- *ip_ptr*: Pointer to previously created IP instance

Return Values

- **NX_SUCCESS** (0x00) ICMP services are successfully enabled
- **NX_PTR_ERROR** (0x07) Invalid IP pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
/* Enable ICMP on the IP instance. */
status = nxd_icmp_enable(&ip_0);

/* A status return of NX_SUCCESS indicates that the IP instance is
   enabled for ICMP services. */
```

See Also

- nx_icmp_enable
- nx_icmp_info_get

- nx_icmp_ping
- nxd_icmp_ping
- nxd_icmp_source_ping
- nxd_icmpv6_ra_flag_callback_set

nxd_icmp_ping

Perform ICMPv4 or ICMPv6 Echo Requests

Prototype

```
UINT nxd_icmp_ping(
    NX_IP *ip_ptr,
    NXD_ADDRESS *ip_address,
    CHAR *data_ptr,
    ULONG data_size,
    NX_PACKET **response_ptr,
    ULONG wait_option);
```

Description

This service sends out an ICMP Echo Request packet through an appropriate physical interface and waits for an Echo Reply from the destination host. NetX Duo determines the appropriate interface, based on the destination address, to send the ping message . Applications shall use the service ***nxd_icmp_source_ping*** to specify the physical interface and precise source IP address to use for packet transmission.

The IP instance must have been created, and the ICMPv4/ICMPv6 services must be enabled (see ***nxd_icmp_enable***).

> **Warning:** If NX_SUCCESS is returned, the application is responsible for releasing the received packet after it is no longer needed.

Parameters

- *ip_ptr*: Pointer to IP instance
- *ip_address*: Destination IP address to ping, in host byte order
- *data_ptr*: Pointer to ping packet data area
- *data_size*: Number of bytes of ping data
- *response_ptr*: Pointer to response packet Pointer
- *wait_option*: Time to wait for a reply. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **timeout value in ticks** (0x00000001 through
  - **NX_WAIT_FOREVER** 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful sent and received ping
- **NX_NOT_SUPPORTED** (0x4B) IPv6 is not enabled
- **NX_OVERFLOW** (0x03) Ping data exceeds packet payload
- **NX_NO_RESPONSE** (0x29) Destination host did not respond
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by tx_thread_wait_abort
- **NX_NO_INTERFACE_ADDRESS** (0x50) No suitable outgoing interface can be found.
- **NX_PTR_ERROR** (0x07) Invalid IP or response pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_NOT_ENABLED** (0x14) IP or ICMP component is not enabled
- **NX_IP_ADDRESS_ERROR** (0x21) Input IP address is invalid

Allowed From

Threads

Preemption Possible

No

Example

```
/* The following two examples illustrate how to use this API to send
   ping packets to IPv6 or IPv4 destinations. */

/* The first example: Send a ping packet to an IPv6 host
   2001:1234:5678::1 */

/* Declare variable address to hold the destination address. */
NXD_ADDRESS ip_address;

char *buffer = "abcd";
UINT prefix_length = 10;

/* Set the IPv6 address. */
ip_address.nxd_ip_address_version  = NX_IP_VERSION_V6;
ip_address.nxd_ip_address.v6[0]    = 0x20011234;
ip_address.nxd_ip_address.v6[1]    = 0x56780000;
ip_address.nxd_ip_address.v6[2]    = 0;
ip_address.nxd_ip_address.v6[3]    = 1;

status = nxd_icmp_ping(&ip_0, &ip_address, buffer,
                       strlen(buffer),&response_ptr,
                       NX_WAIT_FOREVER);
```

```
/* A return value of NX_SUCCESS indicates a ping reply has been
   received from IP address 2001:1234:5678::1 and the response
   packet is contained in the packet pointed to by response_ptr.It
   should have the same "abcd" four bytes of data. */

/* The second example: Send a ping packet to an IPv4 host 1.2.3.4 */

/* Program the IPv4 address. */
ip_address.nxd_ip_address_version  = NX_IP_VERSION_V4;
ip_address.nxd_ip_address.v4[0]    = 0x01020304;

status = nxd_icmp_ping(&ip_0, &ip_address, buffer,
                       strlen(buffer),&response_ptr, 10);

/* A return value of NX_SUCCESS indicates a ping reply was received
   from IP address 1.2.3.4 and the response packet is contained in
   the packet pointed to by response_ptr. It should have the same
   "abcd" four bytes of data. */
```

See Also

- nx_icmp_enable
- nx_icmp_info_get
- nx_icmp_ping
- nxd_icmp_enable
- nxd_icmp_source_ping
- nxd_icmpv6_ra_flag_callback_set

nxd_icmp_source_ping

Perform ICMPv4 or ICMPv6 Echo Requests

Prototype

```
UINT nxd_icmp_source_ping(
    NX_IP *ip_ptr,
    NXD_ADDRESS *ip_address,
    UINT address_index,
    CHAR *data_ptr,
    ULONG data_size,
    NX_PACKET **response_ptr,
    ULONG wait_option);
```

Description

This service sends out an ICMP Echo Request packet using the specified index of an IPv4 or IPv6 address, and through the network interface the source address is associated with, and waits for an Echo Reply from the destination host. This service works with both IPv4 and IPv6 addresses. The parameter *address_index* indicates the source IPv4 or IPv6 address to use. For IPv4 address, the *address_index* is the same index to the attached network interface. For IPv6, the *address_index* indicates the entry in the IPv6 address table.

The IP instance must have been created, and the ICMPv4 and ICMPv6 services must be enabled (see *nxd_icmp_enable*).

> **Caution:** *If NX_SUCCESS is returned, the application is responsible for releasing the received packet after it is no longer needed.*

Parameters

- *ip_ptr*: Pointer to IP instance
- *ip_address*: Destination IP address to ping, in host byte order
- *address_index*: Indicates the IP address to use as source address
- *data_ptr*: Pointer to ping packet data area
- *data_size*: Number of bytes of ping data
- *response_ptr*: Pointer to response packet pointer
- *wait_option*: Time to wait for a reply. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE
  - **NX_WAIT_FOREVER** 0xFFFFFFFF)

Return Values

- **NX_SUCCESS** (0x00) Successful sent and received ping
- **NX_NOT_SUPPORTED** (0x4B) IPv6 is not enabled
- **NX_OVERFLOW** (0x03) Ping data exceeds packet payload
- **NX_NO_RESPONSE** (0x29) Destination host did not respond
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by tx_thread_wait_abort
- **NX_NO_INTERFACE_ADDRESS** (0x50) No suitable outgoing interface can be found
- **NX_PTR_ERROR** (0x07) Invalid IP or response pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_NOT_ENABLED** (0x14) IP or ICMP component is not enabled
- **NX_IP_ADDRESS_ERROR** (0x21) Input IP address is invalid

Allowed From

Threads

Preemption Possible

No

Example

```
/* The following two examples illustrate how to use this API to send ping
   packets to IPv6 or IPv4 destinations. */

/* The first example: Send a ping packet to an IPv6 host
   FE80::411:7B23:40dc:f181 */

/* Declare variable address to hold the destination address. */

#define PRIMARY_INTERFACE 0
#define GLOBAL_IPv6_ADDRESS 1

NXD_ADDRESS ip_address;
char *buffer = "abcd";
UINT prefix_length = 10;

/* Set the IPv6 address. */
ip_address.nxd_ip_address_version  = NX_IP_VERSION_V6;
ip_address.nxd_ip_address.v6[0]    = 0xFE800000;
ip_address.nxd_ip_address.v6[1]    = 0x00000000;
ip_address.nxd_ip_address.v6[2]    = 0x04117B23;
ip_address.nxd_ip_address.v6[3]    = 0x40DCF181;

status = nxd_icmp_source_ping(&ip_0, &ip_address,
                              GLOBAL_IPv6_ADDRESS,
                              buffer,
                              strlen(buffer),
                              &response_ptr,
                              NX_WAIT_FOREVER);

/* A return value of NX_SUCCESS indicates a ping reply has been received
   from IP address FE80::411:7B23:40dc:f181 and the response packet is
   contained in the packet pointed to by response_ptr. It should have the
   same "abcd" four bytes of data. */

/* The second example: Send a ping packet to an IPv4 host 1.2.3.4 */

/* Program the IPv4 address. */
ip_address.nxd_ip_address_version  = NX_IP_VERSION_V4;
ip_address.nxd_ip_address.v4       = 0x01020304;
```

```
status = nxd_icmp_source_ping(&ip_0, &ip_address,
                              PRIMARY_INTERFACE,
                              buffer,
                              strlen(buffer),
                              &response_ptr,
                              NX_WAIT_FOREVER);

/* A return value of NX_SUCCESS indicates a ping reply was received from
   IP address 1.2.3.4 and the response packet is contained in the packet
   pointed to by response_ptr. It should have the same "abcd" four bytes
   of data. */
```

See Also

- nx_icmp_enable
- nx_icmp_info_get
- nx_icmp_ping
- nxd_icmp_enable
- nxd_icmp_ping
- nxd_icmpv6_ra_flag_callback_set

nxd_icmpv6_ra_flag_callback_set

Set the ICMPv6 RA flag change callback function

Prototype

```
UINT nxd_icmpv6_ra_flag_callback_set(
    NX_IP *ip_ptr,
    VOID(*ra_callback)(NX_IP*ip_ptr, UINT ra_flag));
```

Description

This service sets the ICMPv6 Router Advertisement flag change callback function. The user-supplied callback function is invoked when NetX Duo receives a router advertisement message.

Parameters

- *ip_ptr*: Pointer to IP instance
- *ra_callback*: User-supplied callback function

Return Values

- **NX_SUCCESS** (0x00) Successful set the RA flag callback function
- **NX_NOT_SUPPORTED** (0x4B) IPv6 is not enabled
- **NX_PTR_ERROR** (0x07) Invalid IP

- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
VOID icmpv6_ra_flag_callback(NX_IP *ip_ptr, UINT ra_flag)
{
   /* RA flag has changed. The updated value is passed in via the
      ra_flag parameter. */
}

/* Configure the user-defined ICMPv6 RA flag change callback
   function. */
status = nxd_icmpv6_ra_flag_callback_set(&ip_0,
                                         icmpv6_ra_flag_callback);

/* A status return of NX_SUCCESS indicates the callback function has
   been successfully configured. */
```

See Also

- nx_icmp_enable
- nx_icmp_info_get
- nx_icmp_ping
- nxd_icmp_enable
- nxd_icmp_ping
- nxd_icmp_source_ping

nxd_ip_raw_packet_send

Send Raw IP Packet

Prototype

```
UINT nxd_ip_raw_packet_send(
    NX_IP *ip_ptr,
    NX_PACKET *packet_ptr,
    NXD_ADDRESS *destination
    ULONG protocol,
    UINT ttl,
    ULONG tos);
```

Description

This service sends a raw IPv4 or IPv6 packet (no transport-layer protocol headers). On a multihome system, if the system is unable to determine an appropriate interface (for example, if the destination IP address is IPv4 broadcast, multicast or IPv6 multicast address), the primary device is selected. The service ***nxd_ip_raw_packet_source_send*** can be used to specify an outgoing interface. The NetX equivalent is ***nx_ip_raw_packet_send.***

The IP instance must be previously created and raw IP packet handling must be enabled using the ***nx_ip_raw_packet_enable*** service.

> **Warning:** *Unless an error is returned, the application should not release the packet after this call. Doing so will cause unpredictable results because the network driver will also try to release the packet after transmission.*

Parameters

- *ip_ptr*: Pointer to the previously created IP instance
- *packet_ptr*: Pointer to packet to transmit
- *destination_ip*: Pointer to destination address
- *protocol*: Packet protocol stored to the IP header
- *ttl*: Value for TTL or hop limit
- *tos*: Value for TOS or traffic class and flow label

Return Value

- **NX_SUCCESS** (0x00) Raw IP packet successfully sent
- **NX_NO_INTERFACE_ADDRESS** (0x50) No suitable outgoing interface can be found
- **NX_NOT_ENABLED** (0x14) Raw IP handling not enabled
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IPv4 or IPv6 address
- **NX_UNDERFLOW** (0x02) Not enough room for IPv4 or IPv6 header in the packet
- **NX_OVERFLOW** (0x03) Packet append pointer is invalid
- **NX_PTR_ERROR** (0x07) Invalid IP pointer or packet pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_INVALID_PARAMETERS** (0x4D) Not valid IPv6 address input

Allowed From

Threads

Preemption Possible

No

Example

```
NXD_ADDRESS dest_address;

/* Set the destination address,in this case an IPv6 address. */
dest_address.nxd_ip_address_version  = NX_IP_VERSION_V6;
dest_address.nxd_ip_address.v6[0]    = 0x20011234;
dest_address.nxd_ip_address.v6[1]    = 0x56780000;
dest_address.nxd_ip_address.v6[2]    = 0;
dest_address.nxd_ip_address.v6[3]    = 1;

UINT ttl, tos;

ttl = 128;

tos = 0;

/* Enable RAW IP handling on the previously created IP instance.*/
status = nx_raw_ip_packet_enable(&ip_0);

/* Allocate a packet pointed to by packet_ptr from the IP packet
   pool. */
/* Then transmit the packet to the destination address. */

status = nxd_ip_raw_packet_send(&ip_0, packet_ptr, dest_address,
                                NX_PROTOCOL_UDP, ttl, tos);

/* A status return of NX_SUCCESS indicates the packet was
   successfully transmitted. */
```

See Also

- nx_ip_raw_packet_disable
- nx_ip_raw_packet_enable
- nx_ip_raw_packet_filter_set
- nx_ip_raw_packet_receive
- nx_ip_raw_packet_send
- nx_ip_raw_packet_source_send
- nx_ip_raw_receive_queue_max_set
- nxd_ip_raw_packet_source_send

nxd_ip_raw_packet_source_send

Send raw packet using specified source address

Prototype

```
UINT nxd_ip_raw_packet_source_send(
    NX_IP *ip_ptr,
    NX_PACKET *packet_ptr,
    NXD_ADDRESS *destination_ip,
    UINT address_index,
    ULONG protocol,
    UINT ttl,
    ULONG tos);
```

Description

This service sends a raw IPv4 or IPv6 packet using the specified IPv4 or IPv6 address as source address. This service is typically used on a multihome system, if the system is unable to determine an appropriate interface (for example, if the destination IP address is IPv4 broadcast, multicast or IPv6 multicast address). The parameter *address_index* allows the application to specify the source address to use when sending this raw packet.

The IP instance must be previously created and raw IP packet handling must be enabled using the ***nx_ip_raw_packet_enable**: service.

> **Warning:** *Unless an error is returned, the application should not release the packet after this call. Doing so will cause unpredictable results because the network driver will also try to release the packet after transmission.*

Parameters

- *ip_ptr*: IP instance pointer
- *packet_ptr*: Pointer to packet to send
- *destination_ip*: Destination IP address
- *address_index*: Index to the IPv4 or IPv6 addresses to use as source address.
- *protocol*: Value for the protocol field
- *ttl*: Value for ttl or hop limit
- *tos*: Value for tos or traffic class and flow label

Return Values

- **NX_SUCCESS** (0x00) Packet is sent successfully
- **NX_UNDERFLOW** (0x02) Not enough room for IPv4 or IPv6 header in the packet
- **NX_OVERFLOW** (0x03) Packet append pointer is invalid
- **NX_PTR_ERROR** (0x07) Invalid pointer to IP control block, packet, or destination_ip
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

- **NX_NOT_ENABLED** (0x14) Raw processing not enabled
- **NX_IP_ADDRESS_ERROR** (0x21) Address error
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index
- **NX_INVALID_PARAMETERS** (0x4D) Not valid IPv6 address input

Allowed From

Thread

Preemption Possible

No

Example

```
#define SOURCE_ADDRESS_INDEX 2

/* Send a raw packet from specified interface. */
status = nxd_ip_raw_packet_source_send(&ip_0, packet_ptr,
                                       dest_ip,
                                       SOURCE_ADDRESS_INDEX,
                                       NX_IP_RAW,
                                       NX_IP_TIME_TO_LIVE,
                                       NX_IP_NORMAL);

/* If status == NX_SUCCESS, raw packet has been sent out on the
   specified interface. */
```

See Also

- nx_ip_raw_packet_disable
- nx_ip_raw_packet_enable
- nx_ip_raw_packet_filter_set
- nx_ip_raw_packet_receive
- nx_ip_raw_packet_send
- nx_ip_raw_packet_source_send
- nx_ip_raw_receive_queue_max_set
- nxd_ip_raw_packet_send

nxd_ipv6_address_change_notify

Set ipv6 address change notify

Prototype

```
UINT nxd_ipv6_address_change_notify(
    NX_IP *ip_ptr,
    VOID (*ip_address_change_notify)(NX_IP *, UINT, UINT, UINT, ULONG *));
```

Description

This service registers an application callback routine that NetX Duo calls whenever the IPv6 Address is changed.

This service is available if the NetX Duo library is built is the option ***NX_ENABLE_IPV6_ADDRESS_CHANGE_NOTIFY*** defined.

Parameters

- *ip_ptr*: IP control block pointer
- *ip_address_change_notify*: Application callback function

Return Values

- **NX_SUCCESS** (0x00) Successful set
- **NX_NOT_SUPPORTED** (0x4B) IPv6 address change notify feature is not built into the NetX Duo library
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_NOT_ENABLED** (0x14) IPv6 address change notify is not compiled

Allowed From

Thread

Preemption Possible

No

Example

```
VOID ip_address_change_notify(NX_IP *ip_ptr, UINT status,
                              UINT interface_index,
                              UINT address_index,
                              ULONG *ip_address)
{

   /* Do something when ip address changed. */
}

void ip_address_thread_entry(ULONG id)
{

   /* Set the ip address change notify of IP instance. */
   status = nxd_ipv6_address_change_notify (&ip_0, ip_address_change_notify);
```

```
/* If status == NX_SUCCESS, the ip address change notify of IP
   instance was successfully set. */
}
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nxd_ipv6_address_delete

Delete IPv6 Address

Prototype

```
UINT nxd_ipv6_address_delete(
    NX_IP *ip_ptr,
    UINT address_index);
```

Description

This service deletes the IPv6 address at the specified index in the IPv6 address table of the specified IP instance. There is no NetX equivalent.

Parameters

- *ip_ptr*: Pointer to the previously created IP instance
- *address_index*: Index to IP instance IPv6 address table

Return Values

- **NX_SUCCESS** (0x00) Address successfully deleted
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library
- **NX_NO_INTERFACE_ADDRESS** (0x50) No suitable outgoing interface can be found
- **NX_PTR_ERROR** (0x07) Invalid IP pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
NXD_ADDRESS ip_address;
UINT        address_index;

/* Delete the IPv6 address at the specified address table index. */
address_index = 1;
status = nxd_ipv6_address_delete(&ip_0, address_index);

/* A status return of NX_SUCCESS indicates that the IP instance
   address is successfully deleted. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable

- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nxd_ipv6_address_get

Retrieve IPv6 Address and Prefix

Prototype

```
UINT nxd_ipv6_address_get(
    NX_IP *ip_ptr,
    UINT address_index,
    NXD_ADDRESS *ip_address,
    ULONG *prefix_length,
    UINT *interface_index);
```

Description

This service retrieves the IPv6 address and prefix at the specified index in the address table of the specified IP instance. The index of the physical interface the IPv6 address is associated with is returned in the *interface_index* pointer. The NetX equivalent services are ***nx_ip_address_get*** and ***nx_ip_interface_address_get***.

Parameters

- *ip_ptr*: Pointer to the previously created IP instance
- *address_index*: Index to IP instance address table
- *ip_address*: Pointer to the address to set
- *prefix_length*: Length of the address prefix (subnet mask)
- *interface_index*: Pointer to the index of the interface

Return Values

- **NX_SUCCESS** (0x00) IPv6 is successfully enabled
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library.

- **NX_NO_INTERFACE_ADDRESS** (0x50) No interface address, or invalid address_index
- **NX_PTR_ERROR** (0x07) Invalid IP pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
NXD_ADDRESS ip_address;
UINT        address_index;
ULONG       prefix_length;
UINT        interface_index;

/* Get the IPv6 address at the specified address table index. If
   found, the address network interface is returned in the
   interface_index input, as well as the address prefix in the
   prefix_length input. */

address_index = 1;
status = nxd_ipv6_address_get(&ip_0, address_index, &ip_address,
                              &prefix_length, &interface_index);

/* A status return of NX_SUCCESS indicates that the IP instance
   address is successfully retrieved. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get

- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nxd_ipv6_address_set

Set IPv6 Address and Prefix

Prototype

```
UINT nxd_ipv6_address_set(
    NX_IP *ip_ptr,
    UINT interface_index,
    NXD_ADDRESS *ip_address,
    ULONG prefix_length,
    UINT *address_index);
```

Description

This service sets the supplied IPv6 address and prefix to the specified IP instance. If the *address_index* argument is not null, the index into the IPv6 address table where the address is inserted is returned. The NetX equivalent services are ***nx_ip_address_set**: and* **nx_ip_interface_address_set***.

Parameters

- *ip_ptr*: Pointer to the previously created IP instance
- *interface_index*: Index to the interface the IPv6 address is associated with
- *ip_address*: Pointer to the address to set
- *prefix_length*: Length of the address prefix (subnet mask)
- *address_index*: Pointer to the index into the address table where the address is inserted

Return Values

- **NX_SUCCESS** (0x00) IPv6 is successfully enabled
- **NX_NO_MORE_ENTRIES** (0x15) IP address table is full
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library.

- **NX_DUPLICATED_ENTRY** (0x52) The supplied IP address is already used on this IP instance
- **NX_PTR_ERROR** (0x07) Invalid IP pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IPv6 address
- **NX_INVALID_INTERFACE** (0x4C) Interface points to an invalid network interface

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
NXD_ADDRESS ip_address;
UINT        address_index;
UINT        interface_index;

ip_address.nxd_ip_version = NX_IP_VERSION_V6;
ip_address.nxd_ip_address.v6[0] = 0x20010000;
ip_address.nxd_ip_address.v6[1] = 0;
ip_address.nxd_ip_address.v6[2] = 0;
ip_address.nxd_ip_address.v6[3] = 1;

/* First create an IP instance with packet pool, source address, and
   driver.*/
status = nx_ip_create(&ip_0, "NetX IP Instance 0",
                      IP_ADDRESS(1,2,3,4),
                      0xFFFFFF00UL, &pool_0,_nx_ram_network_driver,
                      pointer, 2048, 1);

/* Then enable IPv6 on the IP instance. */
status = nxd_ipv6_enable(&ip_0);

/* Set the IPv6 address (a global address as indicated by the 64 bit
   prefix) using the IPv6 address just created on the primary device
   (index zero). The index into the address table is returned in
   address_index. */
interface_index = 0;
status = nxd_ipv6_address_set(&ip_0, interface_index, &ip_address,
                              64,&address_index);
```

```
/* A status return of NX_SUCCESS indicates that the IPv6 address is
   successfully assigned to the primary interface (interface 0). */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nxd_ipv6_default_router_add

Add an IPv6 Router to Default Router Table

Prototype

```
UINT nxd_ipv6_default_router_add(
    NX_IP *ip_ptr,
    NXD_ADDRESS *router_address,
    ULONG router_lifetime,
    UINT index_index);
```

Description

This service adds an IPv6 default router on the specified physical interface to the default router table. The equivalent NetX IPv4 service is ***nx_ip_gateway_address_set***.

router_address must point to a valid IPv6 address, and the router must be directly accessible from the specified physical interface.

Parameters

- *ip_ptr*: Pointer to previously created IP instance
- *router_address*: Pointer to the default router address, in host byte order
- *router_lifetime*: Default router life time, in seconds. Valid values are:
  - **0xFFFF:** No time out
  - **0-0xFFFE:** Timeout value, in seconds
- *index_index*: Pointer to the valid memory location for the network index index through which the router can be reached

Return Values

- **NX_SUCCESS** (0x00) Default router is successfully added
- **NX_NO_MORE_ENTRIES** (0x17) No more entries available in the default router table.
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IPv6 address
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library.
- **NX_INVALID_PARAMETERS** (0x4D) Not valid IPv6 address input
- **NX_PTR_ERROR** (0x07) Invalid IP instance or storage space
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_INVALID_INTERFACE** (0x4C) Invalid router interface index

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
/* This example adds a default router for the primary interface at
   fe80::1219:B9FF:FE37:ac to the default router table. */

#define PRIMARY_INTERFACE 0

NXD_ADDRESS router_address;

/* Set the router address version to IPv6 */
router_address.nxd_ip_version   = NX_IP_VERSION_V6;

/* Set the IPv6 address, in host byte order. */
```

```
router_address.nxd_ip_address[0] = 0xfe800000;
router_address.nxd_ip_address[1] = 0x0;
router_address.nxd_ip_address[2] = 0x1219B9FF;
router_address.nxd_ip_address[3] = 0xFE3700AC;

/* Set IPv6 default router. */
status = nxd_ipv6_default_router_add(ip_ptr, &router_address,
                                     0xFFFF, PRIMARY_INTERFACE);

/* Unless invalid pointer input is detected by the error checking
   Service, status return is always NX_SUCCESS. */
```

See Also

- nx_ip_gateway_address_clear
- nx_ip_gateway_address_get
- nx_ip_gateway_address_set
- nx_ip_info_get
- nx_ip_static_route_add
- nx_ip_static_route_delete
- nxd_ipv6_default_router_delete
- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_get
- nxd_ipv6_default_router_number_of_entries_get

nxd_ipv6_default_router_delete

Remove IPv6 Router from Default Router Table

Prototype

```
UINT nxd_ipv6_default_router_delete (
    NX_IP *ip_ptr,
    NXD_ADDRESS *router_address);
```

Description

This service deletes an IPv6 default router from the default router table. The equivalent NetX IPv4 service is ***nx_ip_gateway_address_clear***.

Restrictions

The IP instance has been created. *router_address* must point to valid information.

Parameters

- *ip_ptr*: Pointer to a previously created IP instance

- *router_address*: Pointer to the IPv6 default gateway address

Return Values

- **NX_SUCCESS** (0x00) Router successfully deleted
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library.
- **NX_NOT_FOUND** (0x4E) The router entry cannot be found
- **NX_PTR_ERROR** (0x07) Invalid IP instance or storage space
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_INVALID_PARAMETERS** (0x82) Invalid non pointer input

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
/*This example removes a default router:fe80::1219:B9FF:FE37:ac */

NXD_ADDRESS router_address;

/* Set the router_address version to IPv6 */
router_address.nxd_ip_version = NX_IP_VERSION_V6;

/* Program the IPv6 address, in host byte order. */
router_address.nxd_ip_address[0] = 0xfe800000;
router_address.nxd_ip_address[1] = 0x0;
router_address.nxd_ip_address[2] = 0x1219B9FF;
router_address.nxd_ip_address[3] = 0xFE3700AC;

/* Delete the IPv6 default router. */
nxd_ipv6_default_router_delete(ip_ptr, &router_address);
```

See Also

- nx_ip_gateway_address_clearnx_ip_gateway_address_clear
- nx_ip_gateway_address_get
- nx_ip_gateway_address_set
- nx_ip_info_get
- nx_ip_static_route_add
- nx_ip_static_route_delete
- nxd_ipv6_default_router_add

- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_get
- nxd_ipv6_default_router_number_of_entries_getnx_ip_gateway_address_get
- nx_ip_gateway_address_set
- nx_ip_info_get
- nx_ip_static_route_add
- nx_ip_static_route_delete
- nxd_ipv6_default_router_add
- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_get
- nxd_ipv6_default_router_number_of_entries_get

nxd_ipv6_default_router_entry_get

Get default router entry

Prototype

```
UINT nxd_ipv6_default_router_entry_get(
    NX_IP *ip_ptr,
    UINT interface_index,
    UINT entry_index,
    NXD_ADDRESS *router_addr,
    ULONG *router_lifetime,
    ULONG *prefix_length,
    ULONG *configuration_method);
```

Description

This service retrieves a router entry from the default IPv6 routing table that is attached to a specified network device.

Parameters

- *ip_ptr*: IP control block pointer
- *interface_index*: Index of the network interface
- *entry_index*: Entry Index
- *router_addr*: Router IPv6 Address
- *router_lifetime*: Pointer to router life time
- *prefix_length*: Pointer to prefix length
- *configuration_method*: Pointer to the information on how the entry was configured

Return Values

- **NX_SUCCESS** (0x00) Successful get
- **NX_NOT_FOUND** (0x4E) Router entry not found

- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0
NXD_ADDRESS             router_addr;
ULONG                   router_lifetime;
ULONG                   prefix_length;
ULONG                   configuration_method;

/* Get the router entry of specified interface. */
status = nxd_ipv6_default_router_entry_get (&ip_0,
                                            PRIMARY_INTERFACE,
                                            entry_index,
                                            &router_addr,
                                            &router_lifetime,
                                            &prefix_length,
                                            &configuration_method);

/* If status == NX_SUCCESS, the router entry was successfully
   got. */
```

See Also

- nx_ip_gateway_address_clear
- nx_ip_gateway_address_get
- nx_ip_gateway_address_set
- nx_ip_info_get
- nx_ip_static_route_add
- nx_ip_static_route_delete
- nxd_ipv6_default_router_add
- nxd_ipv6_default_router_delete
- nxd_ipv6_default_router_get
- nxd_ipv6_default_router_number_of_entries_get

nxd_ipv6_default_router_get

Retrieve an IPv6 Router from Default Router Table

Prototype

```
UINT nxd_ipv6_default_router_get(
    NX_IP *ip_ptr,
    UINT interface_index
    NXD_ADDRESS *router_address,
    ULONG *router_lifetime,
    ULONG *prefix_length);
```

Description

This service retrieves an IPv6 default router address, lifetime and prefix length on the specified physical interface from the default router table. The equivalent NetX IPv4 service is ***nx_ip_gateway_address_get.***

router_address must point to a valid NXD_ADDRESS structure, so this service can fill in the IPv6 address of the default router.

Parameters

- *ip_ptr*: Pointer to previously created IP instance
- *interface_index*: The index to the network interface that the router is accessible through
- *router_address*: Pointer to the storage space for the return value of the default router address, in host byte order.
- *router_lifetime*: Pointer to the router lifetime
- *prefix_length*: Pointer to the router address prefix length

Return Values

- **NX_SUCCESS** (0x00) Default router is successfully added
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library.
- **NX_NOT_FOUND** (0x4E) Default router not found
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_INVALID_INTERFACE** (0x4C) Invalid router interface index
- **NX_PTR_ERROR** (0x07) Invalid IP instance or storage space

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
/* This example retrieves a default router for the primary device
   from the default router table. */

#define PRIMARY_INTERFACE 0

NXD_ADDRESS router_address;
ULONG       router_lifetime;
ULONG       prefix_length;

/* Get IPv6 default router. */
status = nxd_ipv6_default_router_get(ip_ptr, PRIMARY_INTERFACE,
                                     &router_address,
                                     &router_lifetime,
                                     &prefix_length);

/* If status returns NX_SUCCESS, the router address and related
   information is returned successfully. */
```

See Also

- nx_ip_gateway_address_clear
- nx_ip_gateway_address_get
- nx_ip_gateway_address_set
- nx_ip_info_get
- nx_ip_static_route_add
- nx_ip_static_route_delete
- nxd_ipv6_default_router_add
- nxd_ipv6_default_router_delete
- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_number_of_entries_get

nxd_ipv6_default_router_number_of_entries_get

Get number of default IPv6 routers

Prototype

```
UINT nxd_ipv6_default_router_number_of_entries_get(
    NX_IP *ip_ptr,
    UINT interface_index,
    UINT *num_entries);
```

Description

This service retrieves the number of IPv6 default routers configured on a given network interface.

Parameters

- *ip_ptr*: IP control block pointer
- *interface_index*: Index of the network interface
- *num_entries*: Destination for number of IPv6 routers on a specified network device

Return Values

- **NX_SUCCESS** (0x00) Successful get
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library.
- **NX_INVALID_INTERFACE** (0x4C) Device index value is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer or num_entries pointer

Allowed From

Thread

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0
UINT                    num_entries;

/* Get the router entries of specified interface. */
status = nxd_ipv6_default_router_number_of_entries_get(&ip_0,
                                                       PRIMARY_INTERFACE,
                                                       &num_entries);

/* If status == NX_SUCCESS, the router entries was successfully
   retrieved. */
```

See Also

- nx_ip_gateway_address_clear
- nx_ip_gateway_address_get
- nx_ip_gateway_address_set
- nx_ip_info_get

- nx_ip_static_route_add
- nx_ip_static_route_delete
- nxd_ipv6_default_router_add
- nxd_ipv6_default_router_delete
- nxd_ipv6_default_router_entry_get
- nxd_ipv6_default_router_get

nxd_ipv6_disable

Disable the IPv6 feature

Prototype

```
UINT nxd_ipv6_disable(NX_IP *ip_ptr);
```

Description

This service disables the IPv6 for the specified IP instance. It also clears the default router table, ND cache and IPv6 address table, leaves the all multicast groups, and resets the router solicitation variables. This service has no effect if IPv6 is not enabled.

Parameters

- *ip_ptr*: IP instance pointer

Return Values

- **NX_SUCCESS** (0x00) Successful disable
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library.
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_NOT_SUPPORT** (0x4B) IPv6 module is not compiled
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* Disable IPv6 feature on this IP instance. */
status = nxd_ipv6_disable(&ip_0);
```

```
/* If status == NX_SUCCESS, disables IPv6 feature on IP instance.*/
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nxd_ipv6_enable

Enable IPv6 Services

Prototype

```
UINT nxd_ipv6_enable(NX_IP *ip_ptr);
```

Description

This service enables IPv6 services. When the IPv6 services are enabled, the IP instance joins the all-node multicast group (FF02::1). This service does not set the link local address or global address. Applications should use *nxd_ipv6_address_set* to configure the device network addresses. There is no NetX equivalent.

Parameters

- *ip_ptr*: Pointer to the previously created IP instance

Return Values

- **NX_SUCCESS** (0x00) IPv6 is successfully enabled
- **NX_ALREADY_ENABLED** (0x15) IPv6 is already enabled
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
/* First create an IP instance with packet pool, source address, and
   driver.*/
status = nx_ip_create(&ip_0, "NetX IP Instance 0",
                      IP_ADDRESS(1,2,3,4),
                      0xFFFFFF00UL, &pool_0,_nx_ram_network_driver,
                      pointer, 2048, 1);

/* Then enable IPv6 on the IP instance. */
status = nxd_ipv6_enable(&ip_0);

/* A status return of NX_SUCCESS indicates that the IP instance is
   enabled for IPv6 services. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable

- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_stateless_address_autoconfig_disable
- nxd_ipv6_stateless_address_autoconfig_enable

nxd_ipv6_multicast_interface_join

Join an IPv6 multicast group

Prototype

```
UINT nxd_ipv6_multicast_interface_join(
    NX_IP *ip_ptr,
    NXD_ADDRESS *group_address,
    UINT interface_index);
```

Description

This service allows an application to join a specific IPv6 multicast address on a specific network interface. The link driver is notified to add the multicast address. This service is available if the NetX Duo library is built with the option ***NX_ENABLE_IPV6_MULTICAST*** defined.

Parameters

- *ip_ptr*: IP instance pointer
- *group_address*: IPv6 multicast address
- *interface_index*: The index to the network interface associated with the multicast group

Return Values

- **NX_SUCCESS** (0x00) Successfully enables receiving on IPv6 multicast address
- **NX_NO_MORE_ENTRIES** (0x17) No more entries in the IPv6 multicast table.
- **NX_OVERFLOW** (0x03) No more group addresses available in the device driver

- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature or IPv6 multicast feature is not built into the NetX Duo library
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IPv6 address
- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid

Allowed From

Threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0

/* Join multicast group on this IP instance. */
status = nxd_ipv6_multicast_interface_join(&ip_0,
                                           &group_address,
                                           PRIMARY_INTERFACE);

/* If status == NX_SUCCESS, interface of index on IP instance
   has joined the multicast group. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_getnx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_leave

nxd_ipv6_multicast_interface_leave

Leave an IPv6 multicast group

Prototype

```
UINT nxd_ipv6_multicast_interface_leave(
    NX_IP *ip_ptr,
```

```
    NXD_ADDRESS *group_address,
    UINT interface_index);
```

Description

This service removes the specific IPv6 multicast address from the specific network device. The link driver is also notified of the removal of the IPv6 multicast address.This service is available if the NetX Duo library is built with the option **NX_ENABLE_IPV6_MULTICAST** defined.

Parameters

- *ip_ptr*: IP instance pointer
- *group_address*: IPv6 multicast address
- *interface_index*: The index to the network interface associated with group

Return Values

- **NX_SUCCESS** (0x00) Successful multicast leave
- **NX_ENTRY_NOT_FOUND** (0x16) Entry not found
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature or IPv6 multicast feature is not built into the NetX Duo library
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid IPv6 address
- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid

Allowed From

Threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0

/* Leave multicast address on this IP instance. */
status = nxd_ipv6_multicast_interface_leave(&ip_0,
                                            &group_address,
                                            primary_interface);

/* If status == NX_SUCCESS, interface of index on IP instance
   has left the multicast group. */
```

See Also

- nx_igmp_enable
- nx_igmp_info_get
- nx_igmp_loopback_disable
- nx_igmp_loopback_enable
- nx_igmp_multicast_interface_join
- nx_igmp_multicast_join
- nx_igmp_multicast_interface_leave
- nx_igmp_multicast_leave
- nx_ipv4_multicast_interface_join
- nx_ipv4_multicast_interface_leave
- nxd_ipv6_multicast_interface_join

nxd_ipv6_stateless_address_autoconfig_disable

Disable stateless address autoconfiguration

Prototype

```
UINT nxd_ipv6_stateless_address_autoconfig_disable(
    NX_IP *ip_ptr,
    UINT interface_index);
```

Description

This service disables the IPv6 stateless address auto configuration feature on a specified network device. It has no effect if the IPv6 address has been configured.

This service is available if the NetX Duo library is built with the option **NX_IPV6_STATELESS_AUTOCONFIG_CONTROL** defined.

Parameters

- *ip_ptr*: IP instance pointer
- *interface_index*: The index to the network interface that the IPv6 stateless address autoconfiguration should be disabled.

Return Values

- **NX_SUCCESS** (0x00) Successfully disables stateless address autoconfigure feature.
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature or IPv6 stateless address autoconfig control feature is not built into the NetX Duo library
- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE 0

/* Disable stateless address auto configuration on this IP instance. */
status = nxd_ipv6_stateless_address_autoconfig_disable(&ip_0,
                                                       PRIMARY_INTERFACE);

/* If status == NX_SUCCESS, disables stateless address auto
   configuration on IP instance. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_enable

nxd_ipv6_stateless_address_autoconfig_enable

Enable stateless address autoconfiguration

Prototype

```
UINT nxd_ipv6_stateless_address_autoconfig_enable(
    NX_IP *ip_ptr,
    UINT interface_index);
```

Description

This service enables the IPv6 stateless address auto configuration feature on a specified network device.

This service is available if the NetX Duo library is built with the option **NX_IPV6_STATELESS_AUTOCONFIG_CONTROL** defined.

Parameters

- *ip_ptr*: IP instance pointer
- *interface_index*: The index to the network interface that the IPv6 stateless address autoconfiguration should be enabled.

Return Values

- **NX_SUCCESS** (0x00) Successfully enables stateless address autoconfig feature.
- **NX_ALREADY_ENABLED** (0x15) Already enabled
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature or IPv6 stateless address autoconfig control feature is not built into the NetX Duo library
- **NX_INVALID_INTERFACE** (0x4C) Interface index is not valid
- **NX_PTR_ERROR** (0x07) Invalid IP control block pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
#define PRIMARY_INTERFACE

/* Enable stateless address auto configuration on this
   IP instance. */
```

```
status = nxd_ipv6_stateless_address_autoconfig_enable(&ip_0,
                                                      PRIMARY_INTERFACE);

/* If status == NX_SUCCESS, enables stateless address auto
   configuration on IP instance. */
```

See Also

- nx_ip_auxiliary_packet_pool_set
- nx_ip_address_change_notify
- nx_ip_address_get
- nx_ip_address_set
- nx_ip_create
- nx_ip_delete
- nx_ip_driver_direct_command
- nx_ip_driver_interface_direct_command
- nx_ip_forwarding_disable
- nx_ip_forwarding_enable
- nx_ip_fragment_disable
- nx_ip_fragment_enable
- nx_ip_info_get
- nx_ip_max_payload_size_find
- nx_ip_status_check
- nx_system_initialize
- nxd_ipv6_address_change_notify
- nxd_ipv6_address_delete
- nxd_ipv6_address_get
- nxd_ipv6_address_set
- nxd_ipv6_disable
- nxd_ipv6_enable
- nxd_ipv6_stateless_address_autoconfig_disable

nxd_nd_cache_entry_delete

Delete IPv6 Address entry in the Neighbor Cache

Prototype

```
UINT nxd_nd_cache_entry_delete(
    NX_IP *ip_ptr,
    ULONG *ip_address);
```

Description

This service deletes an IPv6 neighbor discovery cache entry for the supplied IP address. The equivalent NetX IPv4 function is ***nx_arp_static_entry_delete***.

Parameters

- *ip_ptr*: Pointer to previously created IP instance
- *ip_address*: Pointer to IPv6 address to delete, in host byte order

Return Values

- **NX_SUCCESS** (0x00) Successfully deleted the address
- **NX_ENTRY_NOT_FOUND** (0x16) Address not found in the IPv6 neighbor cache
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library
- **NX_PTR_ERROR** (0x07) Invalid IP instance or storage space
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Initialization, threads

Preemption Possible

No

Example

```
/* This example deletes an entry from the neighbor cache table. */

NXD_ADDRESS ip_address;

ip_address.nxd_ip_address_version = NX_IP_VERSION_V6;
ip_address.nxd_ip_address.v6[0]   = 0x20011234;
ip_address.nxd_ip_address.v6[1]   = 0x56780000;
ip_address.nxd_ip_address.v6[2]   = 0;
ip_address.nxd_ip_address.v6[3]   = 1;

/* Delete an entry in the neighbor cache table with the specified IPv6
   address and hardware address. */
status = nxd_nd_cache_entry_delete(&ip_0,
                                   &ip_address.nxd_ip_address.v6[0]);

/* If status == NX_SUCCESS, the entry was deleted from the neighbor cache
   table. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable

- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nxd_nd_cache_entry_set

Add an IPv6 Address/MAC Mapping to Neighbor Cache

Prototype

```
UINT nxd_nd_cache_entry_set(
    NX_IP *ip_ptr,
    NXD_ADDRESS *dest_ip,
    UINT interface_index,
    char *mac);
```

Description

This service adds an entry to the neighbor discovery cache for the specified IP address *ip_address* mapped to the hardware MAC address on the specified network interface index (interface_index). The equivalent NetX IPv4 service is ***nx_arp_static_entry_create***.

Parameters

- *ip_ptr*: Pointer to previously created IP instance
- *dest_ip*: Pointer to IPv6 address instance
- *interface_index*: Index specifying physical network interface where the destination IPv6 address can be reached
- *mac*: Pointer to hardware address.

Return Values

- **NX_SUCCESS** (0x00) Entry successfully added
- **NX_NOT_SUCCESSFUL** (0x43) Invalid cache or no neighbor cache entries available
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library

- **NX_PTR_ERROR** (0x07) Invalid IP instance or storage space
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index value.

Allowed From

Initialization, Threads

Preemption Possible

No

Example

```
/* This example adds an entry on the primary network interface to
   the neighbor cache table. */

#define PRIMARY_INTEFACE 0

NXD_ADDRESS ip_address;
UCHAR hw_address[6] = {0x0, 0xcf,0x01,0x02, 0x03, 0x04};
CHAR  *mac;

mac = (CHAR *)&hw_address[0];

ip_address.nxd_ip_address_version = NX_IP_VERSION_V6;
ip_address.nxd_ip_address.v6[0]   = 0x20011234;
ip_address.nxd_ip_address.v6[1]   = 0x56780000;
ip_address.nxd_ip_address.v6[2]   = 0;
ip_address.nxd_ip_address.v6[3]   = 1;

/* Create an entry in the neighbor cache table with the specified
   IPv6 address and hardware address. */
status = nxd_nd_cache_entry_set(&ip_0,
                                &ip_address.nxd_ip_address.v6[0],
                                PRIMARY_INTERFACE, mac);

/* If status == NX_SUCCESS, the entry was added to the neighbor
   cache table. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send

- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nxd_nd_cache_hardware_address_find

Locate Hardware Address for an IPv6 Address

Prototype

```
UINT nxd_nd_cache_hardware_address_find(
    NX_IP *ip_ptr,
    NXD_ADDRESS *ip_address,
    ULONG *physical_msw,
    ULONG *physical_lsw
    UINT *interface_index);
```

Description

This service attempts to find a physical hardware address in the IPv6 neighbor discovery cache that is associated with the supplied IPv6 address. The index of the network interface through which the neighbor can be reached is also returned in the parameter *interface_index.* The equivalent NetX IPv4 service is ***nx_arp_hardware_address_find***.

Parameters

- *ip_ptr*: Pointer to previously created IP instance
- *ip_address*: Pointer to IP address to find, host byte order
- *physical_msw*: Pointer to the top 16 bits (47-32) of the physical address, in host byte order
- *physical_lsw*: Pointer to the lower 32 bits (31-0) of the physical address in host byte order
- *interface_index*: Pointer to the valid memory location for the interface index specifying the network device on which the IPv6 address can be reached.

Return Values

- **NX_SUCCESS** (0x00) Successfully found the address

- **NX_ENTRY_NOT_FOUND** (0x16) Mapping not in the neighbor cache
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library
- **NX_INVALID_PARAMETERS** (0x4D) The supplied IP address is not version 6.
- **NX_PTR_ERROR** (0x07) Invalid IP instance or storage space
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Threads

Preemption Possible

No

Example

```
/* This example inputs an IP address on the primary network in order
   to obtain the hardware address it is mapped to in the neighbor
   cache able. */

NXD_ADDRESS  ip_address;
ULONG physical_msw, physical_lsw;
UINT  interface_index;

ip_address.nxd_ip_address_version = NX_IP_VERSION_V6;
ip_address.nxd_ip_address.v6[0]   = 0x20011234;
ip_address.nxd_ip_address.v6[1]   = 0x56780000;
ip_address.nxd_ip_address.v6[2]   = 0;
ip_address.nxd_ip_address.v6[3]   = 1;

/* Obtain the hardware address mapped to the supplied global IPv6
   address. */
status = nxd_nd_cache_hardware_address_find(&ip_0, &ip_address,
                                            &physical_msw,
                                            &physical_lsw
                                            &interface_index);

/* If status == NX_SUCCESS, a matching entry was found in the
   neighbor cache table and the hardware address returned in
   variables physical_msw and physical_lsw, the index of the network
   interface through which the neighbor can be reached is stored in
   interface_index. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_invalidate
- nxd_nd_cache_ip_address_find

nxd_nd_cache_invalidate

Invalidate the Neighbor Discovery Cache

Prototype

```
UINT nxd_nd_cache_invalidate(NX_IP *ip_ptr);
```

Description

This service invalidates the entire IPv6 neighbor discovery cache. This service can be invoked either before or after ICMPv6 has been enabled. This service is not applicable to IPv4 connectivity, so there is no NetX equivalent service.

Parameters

- *ip_ptr*: Pointer to IP instance

Return Values

- **NX_SUCCESS** (0x00) Cache successfully invalidated
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library
- **NX_PTR_ERROR** (0x07) Invalid IP instance
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Threads

Preemption Possible

No

Example

```
/* This example invalidates the host neighbor cache table. */

/* Invalidate the cache table bound to the IP instance. */
status = nxd_nd_cache_invalidate (&ip_0);

/* If status == NX_SUCCESS, all entries in the neighbor cache table
   are invalidated. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_ip_address_find

nxd_nd_cache_ip_address_find

Retrieve IPv6 Address for a Physical Address

Prototype

```
UINT nxd_nd_cache_ip_address_find(
    NX_IP *ip_ptr,
    NXD_ADDRESS *ip_address,
    ULONG physical_msw,
    ULONG physical_lsw,
    UINT *interface_index);
```

Description

This service attempts to find an IPv6 address in the IPv6 neighbor discovery cache that is associated with the supplied physical address. The index of the network interface through which the neighbor can be reached is also returned. The equivalent NetX IPv4 service is ***nx_arp_ip_address_find***.

Parameters

- *ip_ptr*: Pointer to previously created IP instance
- *ip_address*: Pointer to valid NXD_ADDRESS structure
- *physical_msw*: Top 16 bits (47-32) of the physical address to find, host byte order
- *physical_lsw*: Lower 32 bits (31-0) of the physical address to find, host byte order
- *interface_index*: Pointer to the network device index through which the IPv6 address can be reached

Return Values

- **NX_SUCCESS** (0x00) Successfully found the address
- **NX_ENTRY_NOT_FOUND** (0x16) Physical address not found in the neighbor cache
- **NX_NOT_SUPPORTED** (0x4B) IPv6 feature is not built into the NetX Duo library
- **NX_PTR_ERROR** (0x07) Invalid IP instance or storage space
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_INVALID_PARAMETERS** (0x4D) MAC address is zero.

Allowed From

Threads

Preemption Possible

No

Example

```
/* This example inputs a hardware address to search on for the
   matching IPv6 global address in the neighbor cache table. */

NXD_ADDRESS ip_address;
ULONG physical_msw = 0xcf;
ULONG physical_lsw = 0x01020304;
UINT  interface_index;

/* Obtain the IPv6 address mapped to the supplied hardware
```

```
   Address on the primary device. */
status = nxd_nd_cache_ip_address_find(&ip_0, &ip_address,
                                      physical_msw, physical_lsw,
                                      &interface_index);

/* If status == NX_SUCCESS, a matching entry was found in the
   neighbor cache table and the global IPv6 address returned in
   variable ip_address. */
```

See Also

- nx_arp_dynamic_entries_invalidate
- nx_arp_dynamic_entry_set
- nx_arp_enable
- nx_arp_entry_delete
- nx_arp_gratuitous_send
- nx_arp_hardware_address_find
- nx_arp_info_get
- nx_arp_ip_address_find
- nx_arp_static_entries_delete
- nx_arp_static_entry_create
- nx_arp_static_entry_delete
- nxd_nd_cache_entry_delete
- nxd_nd_cache_entry_set
- nxd_nd_cache_hardware_address_find
- nxd_nd_cache_invalidate

nxd_tcp_client_socket_connect

Make a TCP Connection

Prototype

```
UINT nxd_tcp_client_socket_connect(
    NX_TCP_SOCKET *socket_ptr
    NXD_ADDRESSS *server_ip,
    UINT server_port,
    ULONG wait_option);
```

Description

This service makes TCP connection using a previously created TCP client socket to the specified server's port. This service works on either IPv4 or IPv6 networks. Valid TCP server ports range from 0 through 0xFFFF. NetX Duo determines the appropriate physical interface based on the server IP address. The NetX IPv4 equivalent is ***nx_tcp_client_socket_connect***.

The socket must have been bound to a local port.

Parameters

- *socket_ptr*: Pointer to previously created TCP socket
- *server_ip*: Pointer to IPv4 or IPv6 destination address, in host byte order
- *server_port*: Server port number to connect to (1 through 0xFFFF), in host byte order
- *wait_option*: Wait option while the connection is being established. The wait options are defined as follows:
  - **NX_NO_WAIT** (0x00000000)
  - **NX_WAIT_FOREVER** (0xFFFFFFFF)
  - **timeout value in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

- **NX_SUCCESS** (0x00) Successful socket connect
- **NX_WAIT_ABORTED** (0x1A) Requested suspension was aborted by a call to tx_thread_wait_abort
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid server IPv4 or IPv6 address
- **NX_NOT_BOUND** (0x24) Socket is not bound
- **NX_NOT_CLOSED** (0x35) Socket is not in a closed state
- **NX_IN_PROGRESS** (0x37) No wait was specified, connection attempt is in progress
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index.
- **NX_NO_INTERFACE_ADDRESS** (0x50) The network interface does not have valid IPv6 address
- **NX_NOT_ENABLED** (0x14) TCP not enabled
- **NX_INVALID_PORT** (0x46) Invalid port
- **NX_PTR_ERROR** (0x07) Invalid socket pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_NOT_CONNECTED** (0x38) Connection fails.

Allowed From

Threads

Preemption Possible

No

Example

```
NXD_ADDRESS peer_ip_address;
ULONG       peer_port;
```

```
/* Set Peer IPv6 address */
peer_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
peer_ip_address.nxd_ip_address.v6[0] = 0x20010000;
peer_ip_address.nxd_ip_address.v6[1] = 0;
peer_ip_address.nxd_ip_address.v6[2] = 0;
peer_ip_address.nxd_ip_address.v6[3] = 0x101;

/* Set peer port number */
peer_port = 2563;

/* Connect to the peer */
status = nxd_tcp_client_socket_connect(socket_ptr,
                                       &peer_ip_address,
                                       peer_port, NX_WAIT_FOREVER);
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_socket_peer_info_get

nxd_tcp_socket_peer_info_get

Retrieves Peer TCP Socket IP Address and Port Number

Prototype

```
UINT nxd_tcp_socket_peer_info_get
    (NX_TCP_SOCKET *socket_ptr,
    NXD_ADDRESS *peer_ip_address,
    ULONG *peer_port);
```

Description

This service retrieves peer IP address and port information for the connected TCP socket over either IPv4 or IPv6 network. The equivalent NetX IPv4 service is ***nx_tcp_socket_peer_info_get***.

Note that *socket_ptr* must point to a TCP socket that is already in the connected state.

Parameters

- *socket_ptr*: Pointer to TCP socket connected to peer host
- *peer_ip_address*: Pointer to IPv4 or IPv6 peer address. The returned IP address is in host byte order.
- *peer_port*: Pointer to peer port number. The returned port number is in host byte order.

Return Values

- **NX_SUCCESS** (0x00) Socket information successfully retrieved
- **NX_NOT_CONNECTED** (0x38) Socket not connected to peer
- **NX_NOT_ENABLED** (0x14) TCP not enabled
- **NX_PTR_ERROR** (0x07) Invalid pointer input
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Threads

Preemption Possible

No

Example

```
NXD_ADDRESS  peer_ip_address;
ULONG        peer_port;

/* Get TCP socket information. */
status = nxd_tcp_socket_peer_info_get(socket_ptr, &peer_ip_address,
                                      &peer_port);
```

```
/* If status == NX_SUCCESS, the service returns valid peer info: */
if(peer_ip_address.nxd_ip_version == NX_IP_VERSION_V4)
    /* Peer IP address is stored in
       peer_ip_address.nxd_ip_address.v4 */

if(peer_ip_address.nxd_ip_version == NX_IP_VERSION_V6)
    /* Peer IP address is stored in
       peer_ip_address.nxd_ip_address.v6 */
```

See Also

- nx_tcp_client_socket_bind
- nx_tcp_client_socket_connect
- nx_tcp_client_socket_port_get
- nx_tcp_client_socket_unbind
- nx_tcp_enable
- nx_tcp_free_port_find
- nx_tcp_info_get
- nx_tcp_server_socket_accept
- nx_tcp_server_socket_listen
- nx_tcp_server_socket_relisten
- nx_tcp_server_socket_unaccept
- nx_tcp_server_socket_unlisten
- nx_tcp_socket_bytes_available
- nx_tcp_socket_create
- nx_tcp_socket_delete
- nx_tcp_socket_disconnect
- nx_tcp_socket_info_get
- nx_tcp_socket_receive
- nx_tcp_socket_receive_queue_max_set
- nx_tcp_socket_send
- nx_tcp_socket_state_wait
- nxd_tcp_client_socket_connect

nxd_udp_packet_info_extract

Extract network parameters from UDP packet

Prototype

```
UINT nxd_udp_packet_info_extract(
    NX_PACKET *packet_ptr,
    NXD_ADDRESS *ip_address,
    UINT *protocol,
    UINT *port,
```

```
    UINT *interface_index);
```

Description

This service extracts network parameters from a packet received on either IPv4 or IPv6 UDP networks. The NetX equivalent service is ***nx_udp_packet_info_extract.***

Parameters

- *packet_ptr*: Pointer to packet.
- *ip_address*: Pointer to sender IP address.
- *protocol*: Pointer to protocol to be returned.
- *port*: Pointer to sender's port number.
- *interface_index*: Pointer to the index of the network interface from which this packet is received

Return Values

- **NX_SUCCESS** (0x00) Packet interface data successfully extracted.
- **NX_INVALID_PACKET** (0x12) Packet is neither IPv4 or IPv6.
- **NX_PTR_ERROR** (0x07) Invalid pointer input
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Threads

Preemption Possible

No

Example

```
/* Extract network data from UDP packet interface.*/
status = nxd_udp_packet_info_extract(packet_ptr, &ip_address,
                                     &protocol, &port,
                                     &interface_index);

/* If status is NX_SUCCESS packet data was successfully retrieved.*/
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind

- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nxd_udp_socket_send

Send a UDP Datagram

Prototype

```
UINT nxd_udp_socket_send(
    NX_UDP_SOCKET *socket_ptr,
    NX_PACKET *packet_ptr,
    NXD_ADDRESS *ip_address,
    UINT port);
```

Description

This service sends a UDP datagram through a previously created and bound UDP socket for either IPv4 or IPv6 networks. NetX Duo finds a suitable local IP address as source address based on the destination IP address. To specify a specific interface and source IP address, the application should use the **nxd_udp_socket_source_send** service.

Note that this service returns immediately regardless of whether the UDP datagram was successfully sent. The NetX (IPv4) equivalent service is ***nx_udp_socket_send***.

The socket must be bound to a local port.

> **Warning:** *Unless an error is returned, the application should not release the packet after this call. Doing so will cause unpredictable results because the network driver will also try to release the packet after transmission.*

Parameters

- *socket_ptr*: Pointer to previously created UDP socket instance
- *packet_ptr*: UDP datagram packet pointer
- *ip_address*: Pointer to destination IPv4 or IPv6 address
- *port*: Valid destination port number between 1 and 0xFFFF), in host byte order

Return Values

- **NX_SUCCESS** (0x00) Successful UDP socket send
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid server IPv4 or IPv6 address
- **NX_NOT_BOUND** (0x24) Socket not bound to any port
- **NX_NO_INTERFACE_ADDRESS** (0x50) No suitable outgoing interface can be found.
- **NX_UNDERFLOW** (0x02) Not enough room for UDP header in the packet
- **NX_OVERFLOW** (0x03) Packet append pointer is invalid
- **NX_PTR_ERROR** (0x07) Invalid socket pointer, address pointer, or packet pointer.
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_NOT_ENABLED** (0x14) UDP has not been enabled
- **NX_INVALID_PORT** (0x46) Port number is not within a valid range

Allowed From

Threads

Preemption Possible

No

Example

```
NXD_ADDRESS ip_address, server_address;

/* Set the UDP Client IPv6 address. */
ip_address.nxd_ip_version = NX_IP_VERSION_V6;
ip_address.nxd_ip_address.v6[0] = 0x20010000;
ip_address.nxd_ip_address.v6[1] = 0;
ip_address.nxd_ip_address.v6[2] = 0;
ip_address.nxd_ip_address.v6[3] = 1;

/* Set the UDP server IPv6 address to send to. */
server_address.nxd_ip_version = NX_IP_VERSION_V6;
server_address.nxd_ip_address.v6[0] = 0x20010000;
```

```
server_address.nxd_ip_address.v6[1] = 0;
server_address.nxd_ip_address.v6[2] = 0;
server_address.nxd_ip_address.v6[3] = 2;

/* Set the global address (indicated by the 64 bit prefix) using the
   IPv6 address just created on the primary device (index 0). We
   don't need the index into the address table, so the last argument
   is set to null. */

interface_index = 0;
status = nxd_ipv6_address_set(&client_ip, interface_index,
                              &ip_address, 64, NX_NULL);

/* Create the UDP socket client_socket with the ip_address and
   allocate a packet pointed to by packet_ptr (not shown). */

/* Send a packet to the UDP server at server_address on port 12. */
status = nxd_udp_socket_send(&client_socket, packet_ptr,
                             &server_address, 12);

/* If status == NX_SUCCESS, the UDP host successfully transmitted
   the packet out the UDP socket to the server. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_source_send
- nxd_udp_source_extract

nxd_udp_socket_source_send

Send a UDP Datagram

Prototype

```
UINT nxd_udp_socket_source_send(
    NX_UDP_SOCKET *socket_ptr,
    NX_PACKET *packet_ptr,
    NXD_ADDRESS *ip_address,
    UINT port,
    UINT address_index);
```

Description

This service sends a UDP datagram through a previously created and bound UDP socket for either IPv4 or IPv6 networks. The parameter *address_index* specifies the source IP address to use for the outgoing packet. Note that the function returns immediately regardless of whether the UDP datagram was successfully sent.

The socket must be bound to a local port.

The NetX (IPv4) equivalent service is ***nx_udp_socket_source_send***.

> **Warning:** *Unless an error is returned, the application should not release the packet after this call. Doing so will cause unpredictable results because the network driver will also try to release the packet after transmission.*

Parameters

- *socket_ptr*: Pointer to previously created UDP socket instance
- *packet_ptr*: UDP datagram packet pointer
- *ip_address*: Pointer to destination IPv4 or IPv6 address port Valid destination port number between 1 and 0xFFFF), in host byte order
- *address_index*: Index specifying the source address to use for the packet

Return Values

- **NX_SUCCESS** (0x00) Successful UDP socket send
- **NX_IP_ADDRESS_ERROR** (0x21) Invalid server IPv4 or IPv6 address
- **NX_NOT_BOUND** (0x24) Socket not bound to any port
- **NX_NO_INTERFACE_ADDRESS** (0x50) No suitable outgoing interface can be found.
- **NX_NOT_FOUND** (0x4E) No suitable interface can be found
- **NX_PTR_ERROR** (0x07) Invalid socket pointer, address, or packet pointer.

- **NX_CALLER_ERROR** (0x11) Invalid caller of this service
- **NX_NOT_ENABLED** (0x14) UDP has not been enabled
- **NX_INVALID_PORT** (0x46) Port number is not within valid range.
- **NX_INVALID_INTERFACE** (0x4C) Specified network interface is valid
- **NX_UNDERFLOW** (0x02) Not enough room for UDP header in the packet
- **NX_OVERFLOW** (0x03) Packet append pointer is invalid

Allowed From

Threads

Preemption Possible

No

Example

```
NXD_ADDRESS ip_address, server_address;
UINT address_index;

/* Set the UDP Client IPv6 address. */
ip_address.nxd_ip_version = NX_IP_VERSION_V6;
ip_address.nxd_ip_address.v6[0] = 0x20010000;
ip_address.nxd_ip_address.v6[1] = 0;
ip_address.nxd_ip_address.v6[2] = 0;
ip_address.nxd_ip_address.v6[3] = 1;

/* Set the UDP server IPv6 address to send to. */
server_address.nxd_ip_version = NX_IP_VERSION_V6;
server_address.nxd_ip_address.v6[0] = 0x20010000;
server_address.nxd_ip_address.v6[1] = 0;
server_address.nxd_ip_address.v6[2] = 0;
server_address.nxd_ip_address.v6[3] = 2;

/* Set the global address (indicated by the 64 bit prefix) using the IPv6
   address just created on the primary device (index 0). The address index
   is needed for nxd_udp_socket_source_send. */

status = nxd_ipv6_address_set(&client_ip, 0,
                              &ip_address, 64, &address_index);

/* Create the UDP socket client_socket with the ip_address and
   allocate a packet pointed to by packet_ptr (not shown). */

/* Send a packet to the UDP server at server_address on port 12. */
```

```
status = nxd_udp_socket_source_send(&client_socket, packet_ptr,
                                    &server_address, 12, address_index);

/* If status == NX_SUCCESS, the UDP host successfully transmitted the
   packet out the UDP socket to the server. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create
- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_source_extract

nxd_udp_source_extract

Retrieve UPD Packet Source Information

Prototype

```
UINT nxd_udp_source_extract(
    NX_PACKET *packet_ptr,
    NXD_ADDRESS *ip_address,
    UINT *port);
```

Description

This service extracts the source IP address and port number from a UDP packet received through the host UDP socket. The NetX (IPv4) equivalent is ***nx_udp_source_extract***.

Parameters

- *packet_ptr*: Pointer to received UDP packet
- *ip_address*: Pointer to NXD_ADDRESS structure to store packet source IP address
- *port*: Pointer to UDP socket port number

Return Values

- **NX_SUCCESS** (0x00) Successful source extract
- **NX_INVALID_PACKET** (0x12) Packet is not valid
- **NX_PTR_ERROR** (0x07) Invalid socket pointer
- **NX_CALLER_ERROR** (0x11) Invalid caller of this service

Allowed From

Threads

Preemption Possible

No

Example

```
NXD_ADDRESS ip_address;
UINT        port;

/* Create the UDP socket client_socket and
   allocate the packet pointed to by packet_ptr (not shown). */

/* Extract the IP address and port of the packet received on the UDP
   socket specified in the packet interface. */
status = nxd_udp_source_extract(&packet_ptr, &ip_address, &port);

/* If status == NX_SUCCESS, the source IP address and port of the
   packet received on the UDP socket was successfully extracted. */
```

See Also

- nx_udp_enable
- nx_udp_free_port_find
- nx_udp_info_get
- nx_udp_packet_info_extract
- nx_udp_socket_bind
- nx_udp_socket_bytes_available
- nx_udp_socket_checksum_disable
- nx_udp_socket_checksum_enable
- nx_udp_socket_create

- nx_udp_socket_delete
- nx_udp_socket_info_get
- nx_udp_socket_port_get
- nx_udp_socket_receive
- nx_udp_socket_receive_notify
- nx_udp_socket_send
- nx_udp_socket_source_send
- nx_udp_socket_unbind
- nx_udp_source_extract
- nxd_udp_packet_info_extract
- nxd_udp_socket_send
- nxd_udp_socket_source_send

nx_link_vlan_set

Sets VLAN tag to interface.

Prototype

```
UINT nx_link_vlan_set(NX_IP *ip_ptr, UINT interface_index, UINT vlan_tag)
```

Description

This function sets VLAN tag to interface. VLAN tag is comprised the PCP and VLAN ID, encoded in host byte order. The PCP is the 3 most significant bits and the VLAN ID is the 12 least significant bits. The PCP is used to prioritize the packet and the VLAN ID is used to identify the VLAN.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to set the VLAN tag.
- *vlan_tag*: VLAN tag to set to the interface.

Return Values

- **NX_SUCCESS** (0x00) Successful socket checksum disable.
- **NX_PTR_ERROR** (0x07) Invalid IP instance.
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index.

Preemption Possible

No

Example

```
UINT vlan_tag = 0x810;
UINT interface_index = 0;
```

```
/* Set VLAN tag to interface. */
status = nx_link_vlan_set(&ip_0, interface_index, vlan_tag);
```

See Also

- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_packet_receive_callback_remove
- nx_link_ethernet_header_parse
- nx_link_vlan_interface_create

nx_link_vlan_get

Get VLAN tag from interface.

Prototype

```
UINT nx_link_vlan_get(NX_IP *ip_ptr, UINT interface_index, UINT *vlan_tag)
```

Description

This function gets VLAN tag from interface, VLAN tag is comprised the PCP and VLAN ID, encoded in host byte order. The PCP is the 3 most significant bits and the VLAN ID is the 12 least significant bits. The PCP is used to prioritize the packet and the VLAN ID is used to identify the VLAN.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to get the VLAN tag.
- *vlan_tag*: Pointer to store the VLAN tag.

Return Values

- **NX_SUCCESS** (0x00) Successful socket checksum disable.
- **NX_PTR_ERROR** (0x07) Invalid IP instance.
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index.
- **NX_NOT_FOUND** (0x4E) VLAN tag not found.

Preemption Possible

No

Example

```
UINT vlan_tag;
UINT interface_index = 0;

/* Get VLAN tag from interface. */
status = nx_link_vlan_get(&ip_0, interface_index, &vlan_tag);
```

See Also

- nx_link_vlan_set
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_packet_receive_callback_remove
- nx_link_ethernet_header_parse
- nx_link_vlan_interface_create

nx_link_vlan_clear

Clears VLAN tag from interface.

Prototype

```
UINT nx_link_vlan_clear(NX_IP *ip_ptr, UINT interface_index)
```

Description

This function clears VLAN tag from interface.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to clear the VLAN tag.

Return Values

- **NX_SUCCESS** (0x00) Successful socket checksum disable.
- **NX_PTR_ERROR** (0x07) Invalid IP instance.
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index.

Preemption Possible

No

Example

```
UINT interface_index = 0;

/* Clear VLAN tag from interface. */
status = nx_link_vlan_clear(&ip_0, interface_index);
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_packet_receive_callback_remove
- nx_link_ethernet_header_parse
- nx_link_vlan_interface_create

nx_link_multicast_join

Join a multicast group.

Prototype

```
UINT nx_link_multicast_join(NX_IP *ip_ptr, UINT interface_index,
                            ULONG physical_address_msw, ULONG physical_address_lsw)
```

Description

This function handles the request to join the specified multicast group on a specified network device.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to join the multicast group.
- *physical_address_msw*: Top 16 bits (47-32) of the multicast address to join.
- *physical_address_lsw*: Lower 32 bits (31-0) of the multicast address to join.

Return Values

- **NX_SUCCESS** (0x00) Successful multicast group join.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.

- **NX_PTR_ERROR** (0x07) Invalid IP pointer.

Preemption Possible

No

Example

```
UINT interface_index = 0;
ULONG physical_address_msw = 0x011b;
ULONG physical_address_lsw = 0x19000000;

/* Join a multicast group. */
status = nx_link_multicast_join(&ip_0, interface_index,
                                physical_address_msw, physical_address_lsw);
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_packet_receive_callback_remove
- nx_link_ethernet_header_parse
- nx_link_vlan_interface_create

nx_link_multicast_leave

Leave a multicast group.

Prototype

```
UINT nx_link_multicast_leave(NX_IP *ip_ptr, UINT interface_index,
                             ULONG physical_address_msw, ULONG physical_address_lsw)
```

Description

This function handles the request to leave the specified multicast group on a specified network device.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to leave the multicast group.

- *physical_address_msw*: Top 16 bits (47-32) of the multicast address to leave.
- *physical_address_lsw*: Lower 32 bits (31-0) of the multicast address to leave.

Return Values

- **NX_SUCCESS** (0x00) Successful multicast group leave.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.

Preemption Possible

No

Example

```
UINT interface_index = 0;
ULONG physical_address_msw = 0x011b;
ULONG physical_address_lsw = 0x19000000;

/* Leave a multicast group. */
status = nx_link_multicast_leave(&ip_0, interface_index,
                                 physical_address_msw, physical_address_lsw);
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_packet_receive_callback_remove
- nx_link_ethernet_header_parse
- nx_link_vlan_interface_create

nx_link_ethernet_packet_send

Send an Ethernet packet.

Prototype

```
UINT nx_link_ethernet_packet_send(NX_IP *ip_ptr, UINT interface_index, NX_PACKET *packet_ptr
                                  ULONG physical_address_msw, ULONG physical_address_lsw, UI
```

Description

This function sends out a link packet with layer 3 header already constructed or raw packet. Ethernet header will be added in this function.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to send the packet.
- *packet_ptr*: Pointer to the packet to send.
- *physical_address_msw*: Top 16 bits (47-32) of the destination MAC address.
- *physical_address_lsw*: Lower 32 bits (31-0) of the destination MAC address.
- *packet_type*: Type of the packet to send.

Return Values

- **NX_SUCCESS** (0x00) Successful packet send.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.

Preemption Possible

No

Example

```
UINT interface_index = 0;
ULONG physical_address_msw = 0x011b;
ULONG physical_address_lsw = 0x19000000;
UINT packet_type = NX_PTP_ETHERNET_TYPE;

/* Send an Ethernet packet. */
status = nx_link_ethernet_packet_send(&ip_0, interface_index, packet_ptr,
                                      physical_address_msw, physical_address_lsw, packet_typ
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_packet_receive_callback_remove

- nx_link_ethernet_header_parse

nx_link_raw_packet_send

Send a raw packet.

Prototype

```
UINT nx_link_raw_packet_send(NX_IP *ip_ptr, UINT interface_index, NX_PACKET *packet_ptr);
```

Description

This function sends out a link packet with layer 2 header already constructed or raw packet.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to send the packet.
- *packet_ptr*: Pointer to the packet to send.

Return Values

- **NX_SUCCESS** (0x00) Successful packet send.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.

Preemption Possible

No

Example

```
UINT interface_index = 0;

/* Send a raw packet. */
status = nx_link_raw_packet_send(&ip_0, interface_index, packet_ptr);
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_packet_receive_callback_add

- nx_link_packet_receive_callback_remove
- nx_link_ethernet_header_parse
- nx_link_vlan_interface_create

nx_link_packet_receive_callback_add

Add a packet receive callback.

Prototype

```
UINT nx_link_packet_receive_callback_add(NX_IP *ip_ptr, UINT interface_index, NX_LINK_RECEIV
                                         UINT packet_type, nx_link_packet_receive_callback *
```

Description

This function adds a receive callback function to specified interface. Multiple callbacks callback functions can be added to each interface. They will be invoked one by one until the packet is consumed. Only packet matching registered packet_type will be passed to callback function. NX_LINK_PACKET_TYPE_ALL can be used to handle all types except TCP/IP ones.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to add the callback.
- *queue_ptr*: Pointer to the receive queue.
- *packet_type*: Type of the packet to receive.
- *callback_ptr*: Pointer to the callback function.
- *context*: Pointer to the context.

Return Values

- **NX_SUCCESS** (0x00) Successful packet send.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.

Preemption Possible

No

Example

```
UINT interface_index = 0;
NX_LINK_RECEIVE_QUEUE queue;
UINT packet_type = NX_PTP_ETHERNET_TYPE;
nx_link_packet_receive_callback callback;
```

```
/* Add a packet receive callback. */
status = nx_link_packet_receive_callback_add(&ip_0, interface_index, &queue,
                                             packet_type, callback, NX_NULL);
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_remove
- nx_link_ethernet_header_parse
- nx_link_vlan_interface_create

nx_link_packet_receive_callback_remove

Remove a packet receive callback.

Prototype

```
UINT nx_link_packet_receive_callback_remove(NX_IP *ip_ptr, UINT interface_index, NX_LINK_REC
```

Description

This function removes a receive callback function to specified interface.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_index*: Index of the network interface to remove the callback.
- *queue_ptr*: Pointer to the receive queue.

Return Values

- **NX_SUCCESS** (0x00) Successful packet send.
- **NX_PTR_ERROR** (0x07) Invalid IP pointer.
- **NX_INVALID_INTERFACE** (0x4C) Device index points to an invalid network interface.

Preemption Possible

No

Example

```
UINT interface_index = 0;
NX_LINK_RECEIVE_QUEUE queue;

/* Remove a packet receive callback. */
status = nx_link_packet_receive_callback_remove(&ip_0, interface_index, &queue);
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_ethernet_header_parse
- nx_link_vlan_interface_create

nx_link_ethernet_header_parse

Parse an Ethernet header.

Prototype

```
UINT nx_link_ethernet_header_parse(NX_PACKET *packet_ptr, ULONG *destination_msb, ULONG *des
                                   ULONG *source_msb, ULONG *source_lsb, USHORT *ether_type,
                                   UCHAR *vlan_tag_valid, UINT *header_size)
```

Description

This function parses Ethernet packet and return each file of header.

Parameters

- *packet_ptr*: Pointer to the packet to parse.
- *destination_msb*: Pointer to store the destination MAC address MSB.
- *destination_lsb*: Pointer to store the destination MAC address LSB.
- *source_msb*: Pointer to store the source MAC address MSB.
- *source_lsb*: Pointer to store the source MAC address LSB.
- *ether_type*: Pointer to store the Ethernet type.
- *vlan_tag*: Pointer to store the VLAN tag.
- *vlan_tag_valid*: Pointer to store the VLAN tag valid.
- *header_size*: Pointer to store the header size.

Return Values

- **NX_SUCCESS** (0x00) Successful packet send.

Preemption Possible

No

Example

```
ULONG destination_msb, destination_lsb, source_msb, source_lsb;
USHORT ether_type, vlan_tag;
UCHAR vlan_tag_valid;
UINT header_size;

/* Parse an Ethernet header. */
nx_link_ethernet_header_parse(packet_ptr, &destination_msb, &destination_lsb,
                              &source_msb, &source_lsb, &ether_type, &vlan_tag,
                              &vlan_tag_valid, &header_size);
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_packet_receive_callback_remove
- nx_link_vlan_interface_create

nx_link_vlan_interface_create

Create a VLAN interface.

Prototype

```
UINT nx_link_vlan_interface_create(NX_IP *ip_ptr, CHAR *interface_name, ULONG ip_address, UI
                                   UINT vlan_tag, UINT parent_interface_index, UINT *interfa
```

Description

This function creates a VLAN interface and bind to parent interface. Any packet received from parent interface will be dispatched to right interface according to the match of VLAN ID.

Parameters

- *ip_ptr*: Pointer to previously created IP instance.
- *interface_name*: Name of the interface.
- *ip_address*: IP address of the interface.
- *network_mask*: Network mask of the interface.
- *vlan_tag*: VLAN tag of the interface.
- *parent_interface_index*: Index of the parent interface.
- *interface_index_ptr*: Pointer to store the index of the interface.

Return Values

NX_SUCCESS (0x00) Successful packet send. **NX_DUPLICATED_ENTRY** (0x4D) Interface is duplicated. **NX_NO_MORE_ENTRIES** (0x4F) No more entries. **NX_INVALID_PARAMETERS** (0x47) Invalid parameters.

Preemption Possible

No

Example

```
status = nx_link_vlan_interface_create(&ip_0, "NetX IP Interface 0:2", IP_ADDRESS(0, 0, 0, 0
if (status)
{
   error_counter++;
}
```

See Also

- nx_link_vlan_set
- nx_link_vlan_get
- nx_link_vlan_clear
- nx_link_multicast_join
- nx_link_multicast_leave
- nx_link_ethernet_packet_send
- nx_link_raw_packet_send
- nx_link_packet_receive_callback_add
- nx_link_packet_receive_callback_remove

nx_shaper_create

Create a shaper.

Prototype

```
UINT nx_shaper_create(NX_INTERFACE *interface_ptr, NX_SHAPER_CONTAINER *shaper_container, NX
```

Description

This function creates shaper in shaper container, and connects the shaper container with interface instance.

Parameters

- *interface_ptr*: Pointer to the interface instance.
- *shaper_container*: Pointer to the shaper container.
- *shaper*: Pointer to the shaper.
- shaper_type: Type of the shaper.
- shaper_driver: Pointer to the shaper driver.

Return Values

NX_SUCCESS (0x00) Successful shaper create. **NX_INVALID_PARAMETERS** (0x47) Invalid parameters. **NX_NO_MORE_ENTRIES** (0x4F) No more entries.

Preemption Possible

No

Example

```
UINT shaper_init(NX_INTERFACE *interface_ptr)
{
UINT status;
UCHAR pcp_list[8];
UCHAR queue_id_list[8];

    status = nx_shaper_create(interface_ptr, &shaper_container, &cbs_shaper, NX_SHAPER_TYPE_
    if (status != NX_SUCCESS)
    {
        return NX_FALSE;
    }

    status = nx_shaper_default_mapping_get(interface_ptr, pcp_list, queue_id_list, 8);
    if (status != NX_SUCCESS)
    {
        return NX_FALSE;
    }

    status = nx_shaper_mapping_set(interface_ptr, pcp_list, queue_id_list, 8);

    return status;
}
```

See Also

- nx_shaper_delete
- nx_shaper_current_mapping_get
- nx_shaper_default_mapping_get
- nx_shaper_mapping_set
- nx_shaper_cbs_parameter_set
- nx_shaper_fp_parameter_set
- nx_shaper_tas_parameter_set

nx_shaper_delete

Delete a shaper.

Prototype

```
UINT nx_shaper_delete(NX_INTERFACE *interface_ptr, NX_SHAPER *shaper)
```

Description

This function deletes a shaper from interface instance, unlink the shaper container with IP interface when there is no shaper exists.

Parameters

interface_ptr: Pointer to the interface instance. *shaper*: Pointer to the shaper.

Return Values

NX_SUCCESS (0x00) Successful shaper delete. **NX_INVALID_PARAMETERS** (0x47) Invalid parameters. **NX_ENTRY_NOT_FOUND** (0x4A) Entry not found.

Preemption Possible

No

Example

```
status = nx_shaper_delete(&ip_0, &cbs_shaper);
```

See Also

- nx_shaper_create
- nx_shaper_current_mapping_get
- nx_shaper_default_mapping_get
- nx_shaper_mapping_set
- nx_shaper_cbs_parameter_set

- nx_shaper_fp_parameter_set
- nx_shaper_tas_parameter_set

nx_shaper_current_mapping_get

Get current mapping of shaper.

Prototype

```
UINT nx_shaper_current_mapping_get(NX_INTERFACE *interface_ptr, UCHAR *pcp_list, UCHAR *queu
```

Description

This function gets the current pcp to HW queue mapping config.

Parameters

- *interface_ptr*: Pointer to the interface instance.
- *pcp_list*: Pointer to the pcp list.
- *queue_id_list*: Pointer to the queue id list.
- *list_size*: Size of the list.

Return Values

NX_SUCCESS (0x00) Successfully get mapping. **NX_INVALID_PARAMETERS** (0x47) Invalid parameters. **NX_NOT_SUPPORTED** (0x4B) Not supported. **NX_NOT_SUCCESSFUL** (0x51) Not successful.

Preemption Possible

No

Example

```
status = nx_shaper_current_mapping_get(&ip_0, pcp_list, queue_id_list, 8);
```

See Also

- nx_shaper_create
- nx_shaper_delete
- nx_shaper_default_mapping_get
- nx_shaper_mapping_set
- nx_shaper_cbs_parameter_set
- nx_shaper_fp_parameter_set
- nx_shaper_tas_parameter_set

nx_shaper_default_mapping_get

Get default mapping of shaper.

Prototype

```
UINT nx_shaper_default_mapping_get(NX_INTERFACE *interface_ptr, UCHAR *pcp_list, UCHAR *queu
```

Description

This function gets the default pcp to HW queue mapping config.

Parameters

- *interface_ptr*: Pointer to the interface instance.
- *pcp_list*: Pointer to the pcp list.
- *queue_id_list*: Pointer to the queue id list.
- *list_size*: Size of the list.

Return Values

NX_SUCCESS Successfully get mapping. **NX_INVALID_PARAMETERS** (0x47) Invalid parameters. **NX_NOT_SUPPORTED** (0x4B) Not supported.

Preemption Possible

No

Example

```
UINT shaper_init(NX_INTERFACE *interface_ptr)
{
UINT status;
UCHAR pcp_list[8];
UCHAR queue_id_list[8];

    status = nx_shaper_create(interface_ptr, &shaper_container, &cbs_shaper, NX_SHAPER_TYPE_
    if (status != NX_SUCCESS)
    {
        return NX_FALSE;
    }

    status = nx_shaper_default_mapping_get(interface_ptr, pcp_list, queue_id_list, 8);
    if (status != NX_SUCCESS)
    {
        return NX_FALSE;
    }
```

```
    status = nx_shaper_mapping_set(interface_ptr, pcp_list, queue_id_list, 8);

    return status;
}
```

See Also

- nx_shaper_create
- nx_shaper_delete
- nx_shaper_current_mapping_get
- nx_shaper_mapping_set
- nx_shaper_cbs_parameter_set
- nx_shaper_fp_parameter_set
- nx_shaper_tas_parameter_set

nx_shaper_mapping_set

Set mapping of shaper.

Prototype

```
UINT nx_shaper_mapping_set(NX_INTERFACE *interface_ptr, UCHAR *pcp_list, UCHAR *queue_id_lis
```

Description

This function sets the pcp to HW queue mapping config.

Parameters

- *interface_ptr*: Pointer to the interface instance.
- *pcp_list*: Pointer to the pcp list.
- *queue_id_list*: Pointer to the queue id list.
- *list_size*: Size of the list.

Return Values

NX_SUCCESS Successfully set mapping. **NX_INVALID_PARAMETERS** (0x47) Invalid parameters. **NX_NOT_SUPPORTED** (0x4B) Not supported.

Preemption Possible

No

Example

```
UINT shaper_init(NX_INTERFACE *interface_ptr)
{
UINT status;
```

```
UCHAR pcp_list[8];
UCHAR queue_id_list[8];

    status = nx_shaper_create(interface_ptr, &shaper_container, &cbs_shaper, NX_SHAPER_TYPE_
    if (status != NX_SUCCESS)
    {
        return NX_FALSE;
    }

    status = nx_shaper_default_mapping_get(interface_ptr, pcp_list, queue_id_list, 8);
    if (status != NX_SUCCESS)
    {
        return NX_FALSE;
    }

    status = nx_shaper_mapping_set(interface_ptr, pcp_list, queue_id_list, 8);

    return status;
}
```

See Also

- nx_shaper_create
- nx_shaper_delete
- nx_shaper_current_mapping_get
- nx_shaper_default_mapping_get
- nx_shaper_cbs_parameter_set
- nx_shaper_fp_parameter_set
- nx_shaper_tas_parameter_set

nx_shaper_cbs_parameter_set

Set CBS parameter of shaper.

Prototype

```
UINT nx_shaper_cbs_parameter_set(NX_INTERFACE *interface_ptr, NX_SHAPER_CBS_PARAMETER *cbs_p
```

Description

This function configures the hardware parameters for CBS shaper.

Parameters

- *interface_ptr*: Pointer to the interface instance.
- *cbs_parameter*: Pointer to the CBS parameter.
- *pcp*: PCP value.

Return Values

NX_SUCCESS Successfully set CBS parameter. **NX_NOT_SUPPORTED** (0x4B) Not supported. **NX_NOT_FOUND** (0x4E) Not found. **NX_NOT_SUPPORTED** (0x4B) Not supported.

Preemption Possible

No

Example

```
    status = nx_srp_cbs_config_get(srp_ptr -> talker[index].class_id,
                                   (INT)port_rate,
                                   srp_ptr -> talker[index].interval,
                                   srp_ptr -> talker[index].max_interval_frames,
                                   srp_ptr -> talker[index].max_frame_size,
                                   interface_ptr -> nx_interface_ip_mtu_size,
                                   idle_slope_a,
                                   max_frame_size_a,
                                   &(srp_ptr -> talker[index].cbs_parameters));
    if(status)
        return status;

    printf("cbs parameters: idle slope: %d, send slope: %d, hi credit: %d, low credit: %d\r\
                srp_ptr -> talker[index].cbs_parameters.idle_slope,
                srp_ptr -> talker[index].cbs_parameters.send_slope,
                srp_ptr -> talker[index].cbs_parameters.hi_credit,
                srp_ptr -> talker[index].cbs_parameters.low_credit);
    if(srp_ptr -> talker[index].class_id == NX_SRP_SR_CLASS_A)
        status = nx_shaper_cbs_parameter_set(interface_ptr, &(srp_ptr -> talker[index].cbs_p
    else
        status = nx_shaper_cbs_parameter_set(interface_ptr, &(srp_ptr -> talker[index].cbs_p
```

See Also

- nx_shaper_create
- nx_shaper_delete
- nx_shaper_current_mapping_get
- nx_shaper_default_mapping_get
- nx_shaper_mapping_set
- nx_shaper_fp_parameter_set
- nx_shaper_tas_parameter_set

nx_shaper_fp_parameter_set

Set FP parameter of shaper.

Prototype

```
UINT nx_shaper_fp_parameter_set(NX_INTERFACE *interface_ptr, NX_SHAPER_FP_PARAMETER *fp_para
```

Description

This function sets the frame preemption parameter, when used with other shapers, FP parameter should be set before other shapers.

Parameters

- *interface_ptr*: Pointer to the interface instance.
- *fp_parameter*: Pointer to the FP parameter.

Return Values

NX_SUCCESS Successfully set FP parameter. **NX_NOT_SUCCESSFUL** (0x51) Not successful.

Preemption Possible

No

Example

```
#ifdef FP_ENABLED
    //fp_config
    memset(&fp_config, 0, sizeof(NX_SHAPER_FP_PARAMETER));
    fp_config.verification_enable = 1;
    fp_config.express_guardband_enable = NX_TRUE;
    fp_config.ha = 0;
    fp_config.ra = 0;
    fp_config.express_queue_bitmap = (1 << 3) | (1 << 2);

    status = nx_shaper_fp_parameter_set(interface_ptr, &fp_config);

    if (status != NX_SUCCESS)
    {
        return NX_FALSE;
    }
#endif
```

See Also

- nx_shaper_create
- nx_shaper_delete
- nx_shaper_current_mapping_get
- nx_shaper_default_mapping_get

- nx_shaper_mapping_set
- nx_shaper_cbs_parameter_set
- nx_shaper_tas_parameter_set

nx_shaper_tas_parameter_set

Set TAS parameter of shaper.

Prototype

```
UINT nx_shaper_tas_parameter_set(NX_INTERFACE *interface_ptr, NX_SHAPER_TAS_CONFIG *tas_conf
```

Description

This function configures the hardware parameters for TAS shaper.

Parameters

- *interface_ptr*: Pointer to the interface instance.
- *tas_config*: Pointer to the TAS config.

Return Values

NX_SUCCESS Successfully set TAS parameter. **NX_NOT_FOUND** (0x4E) Not found.

Preemption Possible

No

Example

```
tas_config.base_time = (ULONG64)100 << 32; //100seconds
tas_config.auto_fill_status = NX_SHAPER_TAS_IDLE_CYCLE_AUTO_FILL_DISABLED;
tas_config.cycle_time = 1000000;
tas_config.traffic_count = 2;

tas_config.traffic[0].pcp = 2;
tas_config.traffic[0].time_offset = 0;
tas_config.traffic[0].duration = 500000;
tas_config.traffic[0].traffic_control = NX_SHAPER_TRAFFIC_OPEN;

tas_config.traffic[1].pcp = 0;
tas_config.traffic[1].time_offset = 500000;
tas_config.traffic[1].duration = 500000;
tas_config.traffic[1].traffic_control = NX_SHAPER_TRAFFIC_OPEN;
```

```
status = nx_shaper_tas_parameter_set(interface_ptr, &tas_config);
```

See Also

- nx_shaper_create
- nx_shaper_delete
- nx_shaper_current_mapping_get
- nx_shaper_default_mapping_get
- nx_shaper_mapping_set
- nx_shaper_cbs_parameter_set
- nx_shaper_fp_parameter_set

nx_srp_init

Initialization of SRP.

Prototype

```
UINT nx_srp_init(NX_SRP *srp_ptr, NX_IP *ip_ptr, UINT interface_index, NX_PACKET_POOL *pkt_p
                 VOID *stack_ptr, ULONG stack_size, UINT priority);
```

Description

This function initialize SRP, it initializes MRP, MSRP, MVRP sequencly, and create a thread in MRP initializaton.

Parameters

- *srp_ptr*: Pointer to SRP instance.
- *ip_ptr*: Pointer to IP instance.
- *interface_index*: Index of the network interface to use SRP.
- *pkt_pool_ptr*: pointer to Packet pool.
- *stack_ptr*: pointer to SRP thread Stack.
- *stack_size*: SRP thread Stack size .
- *priority*: SRP thread priority.

Return Values

- **NX_SUCCESS** (0x00) Successful init
- **NX_INVALID_INTERFACE** (0x4C) Invalid interface index
- **NX_PTR_ERROR** (0x07) Invalid IP pointer

Allowed From

Threads

Preemption Possible

No

Example

```
#define SRP_THREAD_PRIORITY 5
#define SRP_INTERFACE 0
NX_SRP nx_srp;
NX_IP ip_0;
NX_PACKET_POOL pool_0;
ULONG            srp_stack[2048 *2 / sizeof(ULONG)];

/* Create the SRP client instance */
nx_srp_init(&nx_srp, &ip_0, SRP_INTERFACE, &pool_0,
                     (UCHAR *)srp_stack, sizeof(srp_stack), SRP_THREAD_PRIORITY);
```

See Also

- nx_srp_talker_start
- nx_srp_talker_stop
- nx_srp_listener_start
- nx_srp_listener_stop

nx_srp_talker_start

Start SRP talker.

Prototype

```
UINT nx_srp_talker_start(NX_SRP *srp_ptr, NX_MSRP_DOMAIN *srp_domain, UCHAR *stream_id, UCHA
                         UINT max_frame_size, UINT max_interval_frames, NX_MRP_EVENT_CALLBAC
```

Description

This function start SRP talker, it sets event callback functions and register domain, Vlan, stream request.

Parameters

- *srp_ptr*: Pointer to SRP instance.
- *event_callback*: callback invoked by application to monitor the SRP process.
- *stream_id*: stream id of talker advertised.

Return Values

- **NX_SUCCESS** (0x00) Successful start
- **NX_INVALID_PARAMETERS** (0x4D) Invalid parameter
- **NX_MSRP_EVENT_NOT_SUPPORTED** (0x06) unsupported event
- **NX_MSRP_ATTRIBUTE_FIND_ERROR** (0x09) not found attribute

Allowed From

Threads

Preemption Possible

No

Example

```
#define SRP_THREAD_PRIORITY 5
#define SRP_INTERFACE 0
NX_SRP nx_srp;
UINT MaxFrameSize = 1300;
UINT MaxIntervalFrames = 1;
UCHAR dest_addr[6] ={0X91,0XE0,0XF0,0X00,0X0E,0X80};
UCHAR stream_id[8] = {0X00,0X11,0X22,0X33,0X44,0X56,0,1};
NX_MSRP_DOMAIN srp_domain = {5,2,2};
UINT srp_event_callback(NX_MRP_PARTICIPANT* participant, NX_MRP_ATTRIBUTE* attribute, UCHAR

    /* start the SRP client */
    status = nx_srp_talker_start(&nx_srp, &srp_domain, stream_id,  dest_addr,
                            MaxFrameSize, MaxIntervalFrames, srp_event_callback);
```

See Also

- nx_srp_init
- nx_srp_talker_stop
- nx_srp_listener_start
- nx_srp_listener_stop

nx_srp_talker_stop

Stop SRP talker.

Prototype

```
UINT nx_srp_talker_stop(NX_SRP *srp_ptr, UCHAR *stream_id, NX_MSRP_DOMAIN *domain)
```

Description

This function stop SRP talker. It withdraw the domain,Vlan,stream request.

Parameters

- *srp_ptr*: Pointer to SRP instance.
- *stream_id*: stream id of talker advertised.
- *domain*: domain of SRP talker.

Return Values

- **NX_SUCCESS** (0x00) Successful stop
- **NX_INVALID_PARAMETERS** (0x4D) Invalid parameter
- **NX_MSRP_EVENT_NOT_SUPPORTED** (0x06) unsupported event
- **NX_MSRP_ATTRIBUTE_FIND_ERROR** (0x09) not found attribute

Allowed From

Threads

Preemption Possible

No

Example

```
NX_SRP nx_srp;
UCHAR stream_id[8] = {0X00,0X11,0X22,0X33,0X44,0X56,0,1};
NX_MSRP_DOMAIN srp_domain = {5,2,2};

nx_srp_talker_stop(&nx_srp,stream_id, &srp_domain );
```

See Also

- nx_srp_init
- nx_srp_talker_start
- nx_srp_listener_start
- nx_srp_listener_stop

nx_srp_listener_start

Start SRP listener.

Prototype

```
UINT nx_srp_listener_start(NX_SRP *srp_ptr, NX_MRP_EVENT_CALLBACK event_callback, UCHAR *str
```

Description

This function start SRP listener. It enables listener and set user date and callback function.

Parameters

- *srp_ptr*: Pointer to SRP instance.
- *event_callback*: callback invoked by application to monitor the SRP process.
- *stream_id*: stream id of listener attached.

Return Values

- **NX_MSRP_SUCCESS** (0x00) Successful listener start

Allowed From

Threads

Preemption Possible

No

Example

```
NX_SRP nx_srp;
UCHAR stream_id[8] = {0X00,0X11,0X22,0X33,0X44,0X56,0,1};
UINT srp_event_callback(NX_MRP_PARTICIPANT* participant, NX_MRP_ATTRIBUTE* attribute, UCHAR

   nx_srp_listener_start(&nx_srp, srp_event_callback, stream_id)
```

See Also

- nx_srp_init
- nx_srp_talker_start
- nx_srp_talker_stop
- nx_srp_listener_stop

nx_srp_listener_stop

Stop SRP listener.

Prototype

```
UINT nx_srp_listener_stop(NX_SRP *srp_ptr, UCHAR *stream_id, NX_MSRP_DOMAIN *domain)
```

Description

This function stop SRP listener. It unregister the domain,Vlan stream attached to talker.

Parameters

- *srp_ptr*: Pointer to SRP instance.
- *stream_id*: Stream id of listener attached to.
- *domain*: Domain of listener attached to.

Return Values

- **NX_SUCCESS** (0x00) Successful stop
- **NX_INVALID_PARAMETERS** (0x4D) Invalid parameter
- **NX_MSRP_EVENT_NOT_SUPPORTED** (0x06) unsupported event
- **NX_MSRP_ATTRIBUTE_FIND_ERROR** (0x09) not found attribute

Allowed From

Threads

Preemption Possible

No

Example

```
NX_SRP nx_srp;
UCHAR stream_id[8] = {0X00,0X11,0X22,0X33,0X44,0X56,0,1};
NX_MSRP_DOMAIN srp_domain = {5,2,2};

nx_srp_listener_stop(&nx_srp,stream_id, &srp_domain );
```

See Also

- nx_srp_init
- nx_srp_talker_start
- nx_srp_talker_stop
- nx_srp_listener_start