HCC Network Driver User Guide

Version 2.70

For use with Network Driver versions 5.04 and above

Date: 15-Jun-2017 15:57

Table of Contents

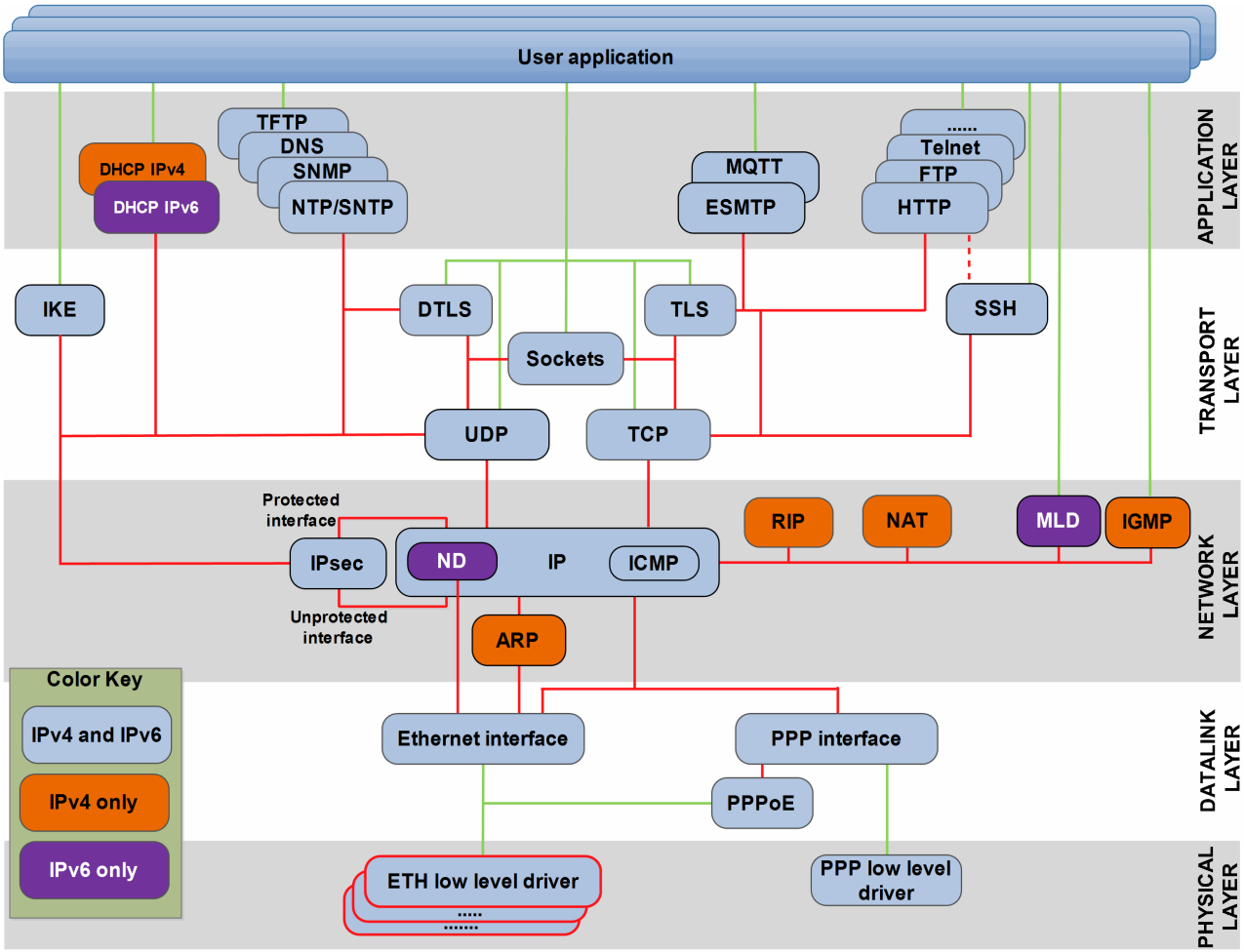
1 System Overview

1.1 Introduction

This guide is for those who want to implement a network driver and describes everything that must be provided by an HCC Embedded network driver.

The network driver design is independent of any particular higher level stack. HCC uses this design for all network drivers and uses it to provide interfaces to its TCP/IP stack and for USB Network driver interfaces.

This diagram shows the driver interface (between the Datalink layer and the Physical layer) within the overall TCP/IP system. (In this diagram green lines show interfaces available to users of the stack, red lines show interfaces internal to the TCP/IP system.)



The network driver system is both highly efficient and very flexible. Its main features include the following:

- You can integrate memory dedicated to your network driver to higher levels of the stack to ensure that zero copy can be used from application to network.
- A network driver can support multiple physical network ports

The basic driver operation is:

1. The initialization function **t_nwdriver_init()** is called to prepare the driver for operation. This call does the following:
   • Provides a set of callback functions for the driver to use when particular events occur.
   • Gets a set of functions to use for communicating with the network driver.
   • Passes a user parameter that is normally used to identify the instance of the network driver being used.
2. The network driver user calls the start function **p_nwfn_start()** before using any other function.
3. Once **p_nwfn_start()** has been called, all other API functions are available for use, except **p_nwfn_delete()**.

The network driver can only be deleted after it has been stopped. After **p_nwfn_stop()** has been called, the driver must ensure that no callbacks are made until **p_nwfn_start()** is called again.

The network driver does not perform buffer management. Note the following:

- **Packet reception** – the network driver user provides buffers for the driver to receive packets into. The user then calls a receive function to get those buffers back with the received packets.
- **Packet transmission** – the network driver user calls the packet send function and the driver transmits the packet then uses the callback to tell the user that transmission is complete and the network driver no longer requires the buffer.

Note: Any network driver that is intended to interoperate with an HCC stack element must conform to this document. To guarantee interoperability, you must use only elements defined within this document. For example, the network driver can only use return codes and status codes as defined here.

1.2 Terminology

This section defines terms used throughout this document. It is important that you understand these terms and their usage.

Term	Description
Network controller	This is typically the module in the microcontroller (or an external module such as an Ethernet controller) that provides the link between the driver and the physical management of the network interface.
Network driver	This is the driver responsible for managing the defined set of network ports that use a single set of physical interfaces to communicate with a network. To take Ethernet as an example, this driver does not handle the Ethernet packet contents. It only handles the management of communicating packets over the physical interface.
Network interface	This is the unique association of a network driver with a single network port.
Network port	A network driver supports one or more network ports. Each port is effectively an instance of the network driver.
Transfer area	This is an area of memory, provided by the user, that is formatted by the Common Interface Layer for packet transmission and reception.
User	The user of the network driver. This is typically a protocol stack that wants to communicate over the network interface(s) provided by the network driver.

1.3 Feature Check

The main features of the network driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Conforms to the HCC Coding Standard including full MISRA compliance.
- Compatible with the HCC IPv4 and IPv6 network stacks.
- Provides a standardized interface for all network drivers.
- Allows multiple instances of each network driver.
- Allows multiple network drivers to be used simultaneously.
- Rich optional feature set including address filtering and checksum offload.
- Memory allocation can be static or dynamic.
- Memory allocation can be set up in two ways:
  - provided by the network driver to the stack
  - made externally and assigned to the network driver.

1.4 Packages and Documents

Packages

The table below lists the packages that you need in order to use the network driver.

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
nw_drv_base	The base network driver package that is required by all network drivers.
nw_drv_xxx_yyy	A specific network driver, where: • **xxx** is the generic driver type (for example, **eth** for Ethernet). • **yyy** is the particular controller (for example, STM32).

Documents

For an overview of the HCC TCP/IP stack software, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC Network Driver User Guide

This is this document.

1.5 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Archive: Network Driver User Guide.
- For the history of changes made to the package code itself, see History: nw_drv_base.

The current version of this manual is 2.70. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
2.70	2017-06-15	5.04	New *Change History* format.
2.60	2017-03-21	5.04	Changes to TCP Stack diagram.
2.50	2017-01-17	5.04	Changes to TCP Stack diagram.
2.40	2016-04-21	5.01	New configuration option, added function group descriptions to API.
2.30	2016-01-05	5.01	Added *Change History* section.
2.20	2015-08-18	5.01	Reorganized *System Overview*.
2.10	2014-02-07	5.01	Various small changes.
2.00	2014-02-06	5.01	First online version.

2 Source File List

This section lists and describes all the source code files included in the system. These files follow HCC Embedded's standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

2.1 API Header File

The file **src/api/api_nwdriver.h** is the only file that must be included by any application using the system. The use of these API functions is defined in Application Programming Interface.

2.2 Version File

The file **src/version/ver_nwdriver.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Application Programming Interface

This section documents the Application Programming Interface (API) of a network driver. It includes all the functions that are available to an application program.

There are two types of function: network driver functions and callback functions.

The network driver API is a set of function pointers. This is so that network drivers and interfaces can be dynamically added to/removed from an HCC protocol stack.

Note:

- All network drivers that want to interoperate with HCC protocol stacks must conform to this API.
- We recommend that the network driver implements all the API functions, but this is not mandatory.
- If an API function is not provided, this limits the functionality of the network stack using that network driver.

3.1 Module Management

The functions are the following:

Function	Description
t_nwdriver_init()	Initializes the network driver for a single network interface; that is, a specific port on a network driver.
p_nwfn_start()	Starts the network driver on the specified network interface.
p_nwfn_stop()	Stops the network driver for this specific network port.
p_nwfn_delete()	Deletes a network interface.

t_nwdriver_init

Use this function to initialize the network driver for a single network interface; that is, a specific port on a network driver.

The network driver structure contains pointers to:

- a t_nwdriver_prop structure that the driver uses to define the network driver properties for the network interface user. This includes, for example, specifying the memory area to be used.
- a t_nwdriver_fn structure of pointers to network driver functions. The network driver must fill this structure with all the network driver functions that it wants to make available to the network interface user.

Format

```
typedef t_nwdriver_ret ( *t_nwdriver_init )(
    const uint32_t          param,
    t_nwdriver * * const    pp_nwdriver )
```

Arguments

Name	Description	Type
param	The driver-specific parameter (for example, a network port).	uint32_t
pp_nwdriver	A pointer to the pointer to the network driver structure.	t_nwdriver * *

Return Values

Return Value	Explanation
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_start

Use this function to start the network driver on the specified network interface.

The callback functions are used by the network driver when the specified events occur within the network driver. The callback function parameter (param) is used as a reference to be included in callback calls. This allows the network interface user to associate the callback with a specific network interface.

> **Note:** No other network driver API function except **p_nwfn_set_hw_addr()** may be called until this start function has been called successfully.

Format

```
t_nwdriver_ret  ( * p_nwfn_start )(
    const t_nwdriver * const          p_nwdriver,
    const t_nwdriver_cb_dsc * const   p_cb_dsc,
    const uint32_t                    param )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver.	t_nwdriver *
p_cb_dsc	A pointer to a structure of pointers to callback functions.	t_nwdriver_cb_dsc *
param	The parameter for the callback functions.	uint32_t

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_stop

Use this function to stop the network driver for this specific network port.

> **Note:** After **p_nwfn_stop()** is called, no API function except **p_nwfn_set_hw_addr()** may be called. The function **p_nwfn_start()** must then be called before other functions are used.

Format

```
t_nwdriver_ret  ( * p_nwfn_stop )( const t_nwdriver * const p_nwdriver )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver.	t_nwdriver *

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_delete

Use this function to delete a network interface.

> **Note:** This function can only be called when the driver has been stopped.

When this function is called, the network driver should free all resources allocated for this interface. The network interface can only be re-established by a new call to **t_nwdriver_init()**.

Format

```
t_nwdriver_ret  ( * p_nwfn_delete )( const t_nwdriver * const p_nwdriver )
```

Arguments

Parameter	Description	Type
p_nwdriver	A pointer to the network driver.	t_nwdriver *

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

3.2 Network Driver Functions

This section describes the following functions:

Function	Description
p_nwfn_receive()	Receives a packet from the network driver.
p_nwfn_send()	Asks the network driver to send a packet.
p_nwfn_add_buf()	Adds a buffer queue of buffers for receiving packets on the specified network interface.
p_nwfn_get_state()	Gets the state of a network interface.
p_nwfn_get_hw_addr()	Gets the hardware address of a network interface.
p_nwfn_set_hw_addr()	Sets the hardware address of a network interface.
p_nwfn_set_filter()	Sets the filtering mode of the network interface.
p_nwfn_set_multicast_table()	Sets up a multicast table.
p_nwfn_get_link_speed()	Gets the speed of the link on the specified network interface in bits per second.

p_nwfn_receive

Use this function to receive a packet from the network driver.

This function is normally called after the network user gets a **p_nwcb_ntf_rx()** receive callback call, though it may also be polled.

Always check the return code to determine whether a packet was made available or not.

> **Note:** You must replenish receive network buffers by using **p_nwfn_add_buf()**. This allows the network driver to continue to receive packets after completed receive packets have been passed back to the user.

Format

```
t_nwdriver_ret  ( * p_nwfn_receive )(
    const t_nwdriver * const   p_nwdriver,
    uint8_t * * const          pp_buf,
    uint16_t * const           p_rlen )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver, the interface to receive on.	t_nwdriver *
pp_buf	Where to write the pointer to the buffer that data was read to.	uint8_t * *
p_rlen	Where to write the number of bytes read.	uint16_t *

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_send

Use this function to ask the network driver to send a packet.

Once the network driver has successfully transmitted the packet, the network interface user receives a p_nwcb_ntf_tx() callback, if these are set up. They can then re-use the buffer as required.

Note:

- This function must be non-blocking.
- The network driver must transmit packets on a network interface in the order that they are received in **p_nwfn_send()** calls to that interface.

Format

```
t_nwdriver_ret  ( * p_nwfn_send )(
    const t_nwdriver * const   p_nwdriver,
    uint8_t * const            p_buf,
    const uint16_t             len )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver, the network interface to use.	t_nwdriver *
p_buf	A pointer to the buffer to be transmitted.	uint8_t *
len	The number of bytes to send.	uint16_t

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_add_buf

Use this function to add a buffer queue of buffers for receiving packets on the specified network interface.

> **Note:** The network driver user must manage the allocation of these buffers to the network driver and its interfaces.

Format

```
t_nwdriver_ret  ( * p_nwfn_add_buf )(
    const t_nwdriver * const   p_nwdriver,
    uint8_t * const            p_buf )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver.	t_nwdriver *
p_buf	A pointer to the buffer. This must be able to store the maximum receive frame size: this is 1536 bytes for Ethernet.	uint8_t *

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_get_state

Use this function to get the state of the specified network interface.

Format

```
t_nwdriver_ret ( * p_nwfn_get_state )(
    const t_nwdriver * const   p_nwdriver,
    t_nwdriver_state * const   p_state )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver.	t_nwdriver *
p_state	Where to write the current state.	t_nwdriver_state *

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_get_hw_addr

Use this function to get the hardware address of the specified network interface.

The hardware address pointed to has length NWDRIVER_HW_ADDRESS_SIZE bytes.

Format

```
t_nwdriver_ret ( * p_nwfn_get_hw_addr )(
    const t_nwdriver * const   p_nwdriver,
    uint8_t * * const          pp_hw_address )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver to query.	t_nwdriver *
pp_hw_address	Where to write the pointer to the hardware address.	uint8_t * *

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_set_hw_addr

Use this function to set the hardware address of the specified network interface.

> **Note:** This function can only be called when the driver is in the stopped state.

The hardware address pointed to has length NWDRIVER_HW_ADDRESS_SIZE bytes.

Format

```
t_nwdriver_ret ( * p_nwfn_set_hw_addr )(
    const t_nwdriver * const   p_nwdriver,
    const uint8_t * const      p_hw_address )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver to change.	t_nwdriver *
p_hw_address	A pointer to the hardware address to set.	uint8_t *

Return Values

Return Value	Explanation
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_set_filter

Use this function to set the filtering mode of the network interface.

This can be very useful when the network controller contains specific logic to filter received packets.

Format

```
t_nwdriver_ret  ( * p_nwfn_set_filter )(
    const t_nwdriver * const   p_nwdriver,
    const uint8_t              mode )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver to apply filtering to.	t_nwdriver *
mode	Filtering mode mask (NWDRIVER_FILTER_XXX), the type of filtering to use.	uint8_t

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_set_multicast_table

Use this function to set up a multicast table.

> **Note:** This function is only useful if NWDRIVER_FILTER_MULTICAST_TBL has been set by using **p_nwfn_set_filter()**. If the table is empty, all multicast packets will be dropped.

Each table entry is of size NWDRIVER_HW_ADDRESS_SIZE.

Format

```
t_nwdriver_ret  ( * p_nwfn_set_multicast_table )(
    const t_nwdriver * const   p_nwdriver,
    const uint8_t * const      p_table,
    const uint16_t             table_size )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver, the network interface to add this table to.	t_nwdriver *
p_table	A pointer to the multicast table.	uint8_t *
table_size	The number of elements in the multicast table.	uint16_t

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

p_nwfn_get_link_speed

Use this function to get the speed of the link on the specified network interface in bits per second.

Format

```
t_nwdriver_ret ( * p_nwfn_get_link_speed )(
    const t_nwdriver * const   p_nwdriver,
    uint32_t * const           p_link_speed )
```

Arguments

Name	Description	Type
p_nwdriver	A pointer to the network driver.	t_nwdriver *
p_link_speed	Where to write the link speed.	uint32_t *

Return Values

Return Value	Description
NWDRIVER_SUCCESS	Successful execution.
Else	See Error Codes.

3.3 Callback Functions

This section describes the following callback functions:

Function	Description
p_nwcb_ntf_rx()	If you provide this function, the network driver calls it whenever a packet is successfully received from the network. It indicates that a packet has been received.
p_nwcb_ntf_tx()	If you provide this function, the network driver calls it whenever a packet is sent and the driver has no further use for it. It indicates that a packet has been sent.
p_nwcb_ntf_state()	Use this function for state change notification, for example when an Ethernet cable is removed from the physical port that the network driver is using.

Note:

- It is the user's responsibility to provide these functions.
- It is recommended that you implement all of these functions, but not mandatory.

p_nwcb_ntf_rx

If you provide this function, the network driver calls it whenever a packet is successfully received from the network. It indicates that a packet has been received.

The recommended action in the callback is to call **p_nwfn_receive()** and to add a new buffer to the network driver to replace the one received.

> **Note:** It is not mandatory to provide this function. The network driver implementation must check that the callback function is not NULL before calling it.

Format

```
void ( * p_nwcb_ntf_rx )(
    uint32_t   param,
    uint16_t   len,
    uint8_t    from_isr )
```

Arguments

Name	Description	Type
param	The network driver parameter.	uint32_t
len	Length of the received frame.	uint16_t
from_isr	Function called from ISR.	uint8_t

p_nwcb_ntf_tx

If you provide this function, the network driver calls it whenever a packet is sent and the driver has no further use for it. It indicates that a packet has been sent.

Packets must be transmitted in the order in which they were received from **p_nwfn_send()** calls.

> **Note:** It is not mandatory to provide this function. The network driver implementation must check that the callback function is not NULL before calling it.

Format

```
void ( * p_nwcb_ntf_tx )(
    uint32_t         param,
    t_nwdriver_ret   ret,
    uint8_t          from_isr )
```

Arguments

Name	Description	Type
param	The network driver parameter.	uint32_t
ret	Completion code.	t_nwdriver_ret
from_isr	Function called from ISR.	uint8_t

p_nwcb_ntf_state

Use this function for state change notification, for example when an Ethernet cable is removed from the physical port that the network driver is using.

If you provide this function, when a change in the connection status of the network port occurs, the network driver calls the function with the parameters you define.

> **Note:** It is not mandatory to provide this function. The network driver implementation must check that the callback function is not NULL before calling it.

Format

```
void ( * p_nwcb_ntf_state )(
    uint32_t   param,
    uint8_t    state,
    uint8_t    from_isr )
```

Arguments

Name	Description	Type
param	The network driver parameter provided at network driver start. This is normally a reference to the network port.	uint32_t
state	New network port state: NWDRIVER_ST_XXX	uint8_t
from_isr	This should be non-zero if this function is called from an ISR, otherwise zero.	uint8_t

3.4 Error Codes

If a function executes successfully, it returns with NWDRIVER_SUCCESS, a value of zero. The following table shows the meaning of the error codes.

Code	Value	Explanation
NWDRIVER_SUCCESS	0	Successful execution.
NWDRIVER_ERROR	1	General error, for example the link is down or has not started.
NWDRIVER_QUEUE_EMPTY	2	Receive queue is empty.
NWDRIVER_INVALID	3	Received packet is invalid.
NWDRIVER_QUEUE_FULL	4	The transmit queue is full. No more packets can be added to it until a packet transmission has completed.

3.5 Types and Definitions

t_nwdriver

The *t_nwdriver* structure is the network driver control structure.

Element	Type	Description
nwd_prop	t_nwdriver_prop	Network driver properties.
p_nwd_fn	t_nwdriver_fn	A pointer to a structure containing network driver functions.
nwd_data	uint32_t	Used by the network driver for internal purposes.

t_nwdriver_cb_dsc

The *t_nwdriver_cb_dsc* structure is a set of callback functions that the user of the network driver provides by using **p_nwfn_start()**. The network driver calls the specified user function when a particular event occurs.

Element	Type	Description
p_nwcb_ntf_state	(* p_nwcb_ntf_state)	The state of the network interface has changed.
p_nwcb_ntf_rx	(* p_nwcb_ntf_rx)	Packet received.
p_nwcb_ntf_tx	(* p_nwcb_ntf_tx)	Packet sent.

t_nwdriver_prop

The *t_nwdriver_prop* structure is provided by the network driver user. It is filled in by the network driver to define its properties for the network driver users.

Element	Type	Description
nwp_options	uint8_t	Option flags (NWDRIVER_OPT_XXX).
nwp_mtu_size	uint16_t	The Maximum Transmission Unit (MTU) size supported by this network interface.
nwp_rxbuf_count	uint16_t	The RX buffer count, used to indicate how many MTU size RX buffers are required by the driver.
p_nwp_buf	uint8_t	The base address of the transfer area, the area receive and transmit buffers are allocated from.
nwp_buf_size	uint32_t	The size of the transfer area.
nwp_header_size	uint8_t	The network driver header size. The driver can request an area preceding the physical frame. This may be used if the driver needs to send an extra header with the frame.
nwp_buf_pad	uint8_t	Buffer pad bytes, the required number of unused bytes to be allocated after the packet data. This may be required if the network controller writes CRC data after the packet.
nwp_buf_align_sh	uint8_t	The required alignment of the packet. This option can be useful if the system can only perform Direct Memory Access (DMA) transfers from an aligned address. Alignment = (1U << nwp_buf_align_sh).
nwp_buf_unit_sh	uint8_t	The unit size shift, required to define packet properties added with **p_nwfn_add_buf()** or sent with **p_nwfn_send()**. All packets must be assigned in a buffer area aligned to the unit size and of a size that is a multiple of the unit size. This does not mean that the len field of **p_nwfn_send()** should be a multiple of unit size, only that the caller must make sure nobody else uses that area during the transfer. This can be used to provide a cached transfer area to the higher layer and be able to call cache invalidate in **p_nwfn_add_buf()** and flush in **p_nwfn_send()**. Unit size = (1U << nwp_buf_unit_sh).

t_nwdriver_fn

The *t_nwdriver_fn* structure provides the complete set of functions available to the user of the network driver.

All these functions are specific to a network interface. That is, a network driver can support multiple physical network interfaces and these functions are called on a specific network interface.

Element	Type	Description
p_nwfn_start	(* p_nwfn_start)	Starts the network interface.
p_nwfn_stop	(* p_nwfn_stop)	Stops the network interface.
p_nwfn_delete	(* p_nwfn_delete)	Deletes the network interface.
p_nwfn_get_state	(* p_nwfn_get_state)	Gets the state of the network interface.
p_nwfn_receive	(* p_nwfn_receive)	Receives a packet.
p_nwfn_send	(* p_nwfn_send)	Sends a packet.
p_nwfn_add_buf	(* p_nwfn_add_buf)	Adds a buffer for packet reception on the network interface.
p_nwfn_get_hw_addr	(* p_nwfn_get_hw_addr)	Gets the hardware address of the network interface.
p_nwfn_set_hw_addr	(* p_nwfn_set_hw_addr)	Sets the hardware address of the network interface.
p_nwfn_set_filter	(* p_nwfn_set_filter)	Sets the filtering method used on the network interface.
p_nwfn_set_multicast_table	(* p_nwfn_set_multicast_table)	Sets the multicast filter table for the network interface.
p_nwfn_get_link_speed	(* p_nwfn_get_link_speed)	Gets the speed of the link on the network interface.

Network Driver Options

These bit field options may be OR'd together. They are set when the network driver initialization function is called by setting the *nwp_options* field of the network driver properties structure.

Code	Default	Description
NWDRIVER_OPT_HW_CHECKSUM	1U	Driver has hardware checksum checking enabled. Set this to 0 to switch it off.
NWDRIVER_OPT_HW_FILTER	2U	Driver has hardware address filtering enabled. Set this to 0 to switch it off.

Network Driver Filtering Types

The type of hardware filtering to be performed by the network driver is set by ORing together the flags below. They are sent to the driver by using the mode argument of the **p_nwfn_set_filter()** function.

Code	Definition
NWDRIVER_FILTER_PERFECT	Accepts packets that match the configured hardware address of this network interface.
NWDRIVER_FILTER_BROADCAST	Accepts packets sent to the network broadcast address
NWDRIVER_FILTER_MULTICAST	Accepts packets sent to any multicast network address.
NWDRIVER_FILTER_MULTICAST_TBL	Accepts packets sent to any multicast address in the configured multicast table.
NWDRIVER_FILTER_PROMISCUOUS	Accepts all packets received from the network.

Network Driver Hardware Address Size

Code	Default	Definition
NWDRIVER_HW_ADDRESS_SIZE	6U	The number of octets in the network address. This is always 6 for Ethernet networks.

Multicast Hardware Address

Code	Default	Definition
NWDRIVER_MULTICAST_HW_SIZE	3U	The multicast hardware size.
NWDRIVER_MULTICAST_HW_IP_MASK	0x007FFFFFU	The mask for multicast hardware.

Network Driver Default MTU Size

Code	Default	Definition
NWDRIVER_DEF_MTU_SIZE	1514U	The network driver's Maximum Transmission Unit (MTU) size.
NWDRIVER_VLAN_HEADER_SIZE	4U	The VLAN header size.

Null and Broadcast Hardware Addresses

Code	Default
NWDRIVER_NULL_HW_ADDRESS	{ 0x00U, 0x00U, 0x00U, 0x00U, 0x00U, 0x00U }
NWDRIVER_BROADCAST_HW_ADDRESS	{ 0xFFU, 0xFFU, 0xFFU, 0xFFU, 0xFFU, 0xFFU }
NWDRIVER_MULTICAST_HW_ADDRESS	{ 0x01U, 0x00U, 0x5EU, 0x00U, 0x00U, 0x00U }

Network States

These are the possible network states that can be returned by a call of the **p_nwfn_get_state()** function.

Code	Definition
NWDRIVER_ST_STOPPED	The network driver has not been started, or has been stopped.
NWDRIVER_ST_CONNECTED	The network driver is connected to the network and is functioning normally.
NWDRIVER_ST_DISCONNECTED	The network driver is not connected to the network.
NWDRIVER_ST_OFFLINE	The network driver is initialized but communication on it is prohibited.

Network Driver Parameters

Code	Default	Description
NWDRIVER_PARAM_VLAN_ENABLE	0x80000000U	If set, this tells the network driver to enable its VLAN capability.
NWDRIVER_PARAM_POS (v)	((v) & 0xFFU)	A mask for the user to clear all bits except those that are user-defined. For example, the network port could be specified in these 8 bits.