FreRTOS TSN Compatibility Layer

Generated by Doxygen 1.9.4

Chapter 1

FreeRTOS TSN Compatibility Layer

The FreeRTOS TSN Compatibility Layer is an additional component working alongside FreeRTOS and its FreeRTOS-Plus-TCP addon. The purpose of this project is to extend the features provided by the Plus TCP addon in order to provide better support over TSN networks. This comprises:

- **The possibility of employing a multi-level queuing scheduler for scheduling packets, completely integrating Plus TCP's Network Event Queue**: instead of using one single queue, with the provided API the user can specify a generalized queue hierarchy starting from simple building blocks. The user can match the packets to queues by assigning filtering policies to each queue.
- **Support for VLAN tagging and Differentiated services**: by tuning the socket options it is possible to enable the insertion of VLAN tags in the Ethernet header and setting the Differentiated services code in the IP header for sent packets
- **Support for control messages over sockets**: this is a mechanism similar to Linux ancillary messages, allowing to add/retrieve additional information to sent/received packets. This also allows the user to generate timestamps for packets, using an API that is similar to the Linux one.

1.1 Setting up the project

This project requires FreeRTOS and FreeRTOS Plus TCP addons, without any modification required. In order to provide the given functionalities, this library should act as an intermediary between FreeRTOS Plus TCP and the Network Interface. In order to do that, we created a wrapper for the network interface that hijacks the traffic towards out implementation. This means that the previous NetworkInterface.c **should not be compiled** with the sources, and NetworkWrapper.c should be compiled in its place. Remember that the chosen NetworkInterface.c should as well be present in the include list, with its dependencies.

If you are having troubles with setting up the project, you can find an example in `this repository` using Makefile.

1.2 Configuration

The project allows configuration at different levels:

1.2.1 Network scheduler

The network scheduler is a tree like structure that manages the order in which packets are served. The root of the tree is where the scheduling starts from, and the leaves of the trees are the queues holding the packets.\ The network scheduler queues should be specified by defining the `vNetworkQueueInit` function in the user project. You can find a template in the `templates/` directory and a working example here.\ The signature of the function is:

```
void vNetworkQueueInit( void );
```

It has the duty of allocating schedulers and queues, linking them and assigning the scheduler root by calling `xNetworkQueueAssignRoot()`.\ Queues are created by calling `pxNetworkQueueCreate()`, and has type `NetworkQueue_t`, schedulers has type `NetworkNode_t` instead, and are created using a functions that are specific for each scheduler.\ If a scheduler admits only one children, it is possibile to link a queue to it using `xNetworkSchedulerLinkQueue()`. To link another scheduler, `xNetworkSchedulerLinkChild()` should be used.\ Please note that it is not possible to link both a scheduler and a queue to the same scheduler. Consider creating a FIFO scheduler before and linking the queue to the FIFO and the FIFO to the previous scheduler.

If the user wants to create his custom schedulers, `FreeRTOS_TSN_NetworkSchedulerBlock.h` provides an useful API that allows to do so. Also check the example in `templates/`.

1.2.2 Timebase

In order to give a better estimate of the timing, the user should specify the timebase that is used for acquiring timestamps.\ The interfacing with the timebase should be defined by the user with the project sources. A function

```
void vTimebaseInit( void )
```

should be defined, creating a `TimebaseHandle_t` object, assigning the required functions and calling `xTimebaseHandleSet()`. Please note that this function is also expected to start the timebase.

You can see an example of configuration for an STM32 board here.

1.2.3 TSN Config

As with FreeRTOS and FreeRTOS-Plus-TCP a configuration file must be provided by the user. You can start from the template in `templates/` directory and find more details for the single parameters in the default configuration file `FreeRTOSTSNConfigDefaults.h`.

1.3 TSN Sockets

Whereas the scheduling functions are shared with sockets in FreeRTOS Plus TCP, some features have required the definition of a socket extension, that we call *TSN Sockets*. The API to use this sockets is the same used for the Plus TCP sockets, but includes these additional capabilities:

- Setting the VLAN tag and DSCP socket options for the packets being sent by this socket
- Enabling timestamping for received or sent packets.
- Using recvmsg() to retrieve a packet together with its ancillary control data.

An example of usage can be found here.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 cmsghdr Struct Reference

`#include <FreeRTOS_TSN_Ancillary.h>`

Data Fields

- socklen_t cmsg_len
- int cmsg_level
- int cmsg_type

4.1.1 Field Documentation

4.1.1.1 cmsg_len

`socklen_t cmsghdr::cmsg_len`

4.1.1.2 cmsg_level

`int cmsghdr::cmsg_level`

4.1.1.3 cmsg_type

`int cmsghdr::cmsg_type`

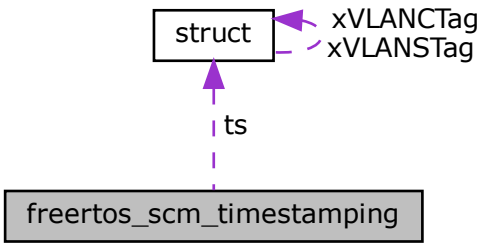
The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_Ancillary.h

4.2 freertos_scm_timestamping Struct Reference

`#include <FreeRTOS_TSN_Timestamp.h>`

Collaboration diagram for freertos_scm_timestamping:



Data Fields

- struct freertos_timespec ts [3]

4.2.1 Field Documentation

4.2.1.1 ts

`struct freertos_timespec freertos_scm_timestamping::ts[3]`

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_Timestamp.h

4.3 freertos_timespec Struct Reference

FreeRTOS implementation of a timespec.

`#include <FreeRTOS_TSN_Timebase.h>`

Data Fields

- uint32_t tv_sec
- uint32_t tv_nsec

4.3.1 Detailed Description

FreeRTOS implementation of a timespec.

Important note: here the tv_sec is an unsigned 32 bit integer. While in the other implementations it is usually a signed type.

4.3.2 Field Documentation

4.3.2.1 tv_nsec

```
uint32_t freertos_timespec::tv_nsec
```

4.3.2.2 tv_sec

```
uint32_t freertos_timespec::tv_sec
```

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_Timebase.h

4.4 iovec Struct Reference

```
#include <FreeRTOS_TSN_Ancillary.h>
```

Data Fields

- void * iov_base
- size_t iov_len

4.4.1 Field Documentation

4.4.1.1 iov_base

```
void* iovec::iov_base
```

4.4.1.2 iov_len

```
size_t iovec::iov_len
```

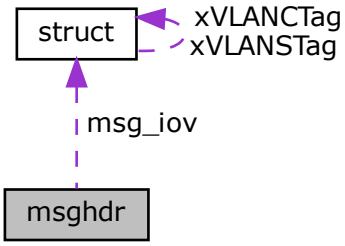
The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_Ancillary.h

4.5 msghdr Struct Reference

```
#include <FreeRTOS_TSN_Ancillary.h>
```

Collaboration diagram for msghdr:



Data Fields

- void * msg_name
- socklen_t msg_namelen
- struct iovec * msg_iov
- size_t msg_iovlen
- void * msg_control
- size_t msg_controllen
- int msg_flags

4.5.1 Field Documentation

4.5.1.1 msg_control

```
void* msghdr::msg_control
```

4.5.1.2 msg_controllen

```
size_t msghdr::msg_controllen
```

4.5.1.3 msg_flags

```
int msghdr::msg_flags
```

4.5.1.4 msg_iov

```
struct iovec* msghdr::msg_iov
```

4.5.1.5 msg_iovlen

```
size_t msghdr::msg_iovlen
```

4.5.1.6 msg_name

```
void* msghdr::msg_name
```

4.5.1.7 msg_namelen

```
socklen_t msghdr::msg_namelen
```

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_Ancillary.h

4.6 sock_extended_err Struct Reference

```
#include <FreeRTOS_TSN_Ancillary.h>
```

Data Fields

- uint32_t ee_errno
- uint8_t ee_origin
- uint8_t ee_type
- uint8_t ee_code
- uint8_t ee_pad
- uint32_t ee_info
- uint32_t ee_data

4.6.1 Field Documentation

4.6.1.1 ee_code

```
uint8_t sock_extended_err::ee_code
```

4.6.1.2 ee_data

```
uint32_t sock_extended_err::ee_data
```

4.6.1.3 ee_errno

```
uint32_t sock_extended_err::ee_errno
```

4.6.1.4 ee_info

```
uint32_t sock_extended_err::ee_info
```

4.6.1.5 ee_origin

```
uint8_t sock_extended_err::ee_origin
```

4.6.1.6 ee_pad

```
uint8_t sock_extended_err::ee_pad
```

4.6.1.7 ee_type

```
uint8_t sock_extended_err::ee_type
```

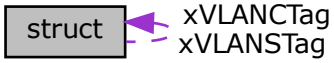
The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_Ancillary.h

4.7 struct Struct Reference

```
#include <FreeRTOS_TSN_VLANTags.h>
```

Collaboration diagram for struct:



Data Fields

- MACAddress_t xDestinationAddress
- MACAddress_t xSourceAddress
- struct xVLAN_TAG xVLANSTag
- struct xVLAN_TAG xVLANCTag
- uint16_t usFrameType

4.7.1 Field Documentation

4.7.1.1 usFrameType

```
uint16_t struct::usFrameType
```

The EtherType field 12 + 2 = 14

4.7.1.2 xDestinationAddress

```
MACAddress_t struct::xDestinationAddress
```

Destination address 0 + 6 = 6

4.7.1.3 xSourceAddress

```
MACAddress_t struct::xSourceAddress
```

Source address 6 + 6 = 12

4.7.1.4 xVLANCTag

```
struct xVLAN_TAG struct::xVLANCTag
```

4.7.1.5 xVLANSTag

```
struct xVLAN_TAG struct::xVLANSTag
```

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_VLANTags.h

4.8 xNETQUEUE Struct Reference

A network queue structure, a leaf in the network scheduler tree.

```
#include <FreeRTOS_TSN_NetworkSchedulerQueue.h>
```

Data Fields

- QueueHandle_t xQueue
- UBaseType_t uxIPV
- eQueuePolicy_t ePolicy
- char cName [tsnconfigMAX_QUEUE_NAME_LEN]
- FilterFunction_t fnFilter

4.8.1 Detailed Description

A network queue structure, a leaf in the network scheduler tree.

This is wrapper to a basic FreeRTOS queue. In addition to that, it contains:

- A queuing policy, that specifies whether this is queue is limited to store outbound and/or inbound packets. eIPTaskEvents is a special flag to hint the network scheduler to reuse the network event queue inside the Plus TCP addon. If the destination is TSN socket this will behave like eSendRecv. Note that unsupported traffic (i.e. ARP, TCP) will always be passed to Plus TCP.
- An internal priority value, this should in thery be the maximum priority of tasks which should receive/send packet on this queue. This is currently used when a packet matches multiple queue policy, so that the queue with the highest IPV is chosen. This field can be used to enable a dynamic priority for the TSN controller tasks, if the respective config entry is enabled: the controller assumes a priority which is equal to the maximum IPV among all the queues which have waiting packets.
- The filter function which restricts the type of packets that this queue is allowed to accept. This must be the signature of FilterFunction_t and return either pdTRUE or pdFALSE.
- The name field is currently unused in the socket API, but it can be used to insert a packet in a specific queue, without letting the scheduler decide on its own.

4.8.2 Field Documentation

4.8.2.1 cName

```
char xNETQUEUE::cName[tsnconfigMAX_QUEUE_NAME_LEN]
```

Name of the queue

4.8.2.2 ePolicy

```
eQueuePolicy_t xNETQUEUE::ePolicy
```

Policy for message direction

4.8.2.3 fnFilter

```
FilterFunction_t xNETQUEUE::fnFilter
```

Function to filter incoming packets

4.8.2.4 uxIPV

```
UBaseType_t xNETQUEUE::uxIPV
```

Internal priority value

4.8.2.5 xQueue

```
QueueHandle_t xNETQUEUE::xQueue
```

FreeRTOS queue handle

The documentation for this struct was generated from the following file:

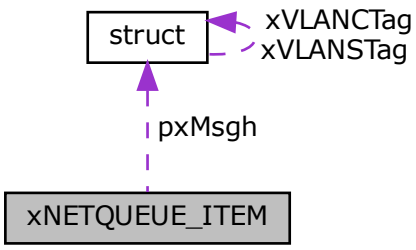
- source/include/FreeRTOS_TSN_NetworkSchedulerQueue.h

4.9 xNETQUEUE_ITEM Struct Reference

The structure used in the network scheduler queues.

```
#include <FreeRTOS_TSN_NetworkSchedulerQueue.h>
```

Collaboration diagram for xNETQUEUE_ITEM:



Data Fields

- eIPEvent_t eEventType
- NetworkBufferDescriptor_t * pxBuf
- struct msghdr * pxMsgh
- BaseType_t xReleaseAfterSend

4.9.1 Detailed Description

The structure used in the network scheduler queues.

eEventType should be either eNetworkTxEvent for transmissions or eNetworkRxEvent for receptions. An additional remark for pxMsgh and pxBuf:

- For receptions, the ancillary message should always be present. This is because we are reusing Plus TCP structures, and the UDP packet list inside the TSN socket will only store one pointer. In the normal sockets this list contains pointers to network buffer descriptors, in TSN sockets it will store pointers to message headers.
- For transmissions, we don't currently support sendmsg() and therefore it is allowed to leave this field as NULL.
- In any case, if we are carrying an ancillary message, its iovec buffer should always point to the network buffer descriptor. When passing the queue item to the Plus TCP functions, pxBuf is rewritten to point to the ancillary msg, and then the original network buffer can be retrieved by accessing the iovec buffer of the msghdr.

```
pxBuf->pucEthernetBuffer == pxMsgh
pxMsgh->msg_iov[ 0 ].iov_base == pucOriginalEtherBuffer
```

4.9.2 Field Documentation

4.9.2.1 eEventType

```
eIPEvent_t xNETQUEUE_ITEM::eEventType
```

Specifies whether this is packet is a transmittion or reception

4.9.2.2 pxBuf

```
NetworkBufferDescriptor_t* xNETQUEUE_ITEM::pxBuf
```

4.9.2.3 pxMsgh

```
struct msghdr* xNETQUEUE_ITEM::pxMsgh
```

Pointer to the network buffer holding the data Pointer to message header holding ancillary data

4.9.2.4 xReleaseAfterSend

```
BaseType_t xNETQUEUE_ITEM::xReleaseAfterSend
```

Boolean specifying whether the network buffer should be released after its usage

The documentation for this struct was generated from the following file:

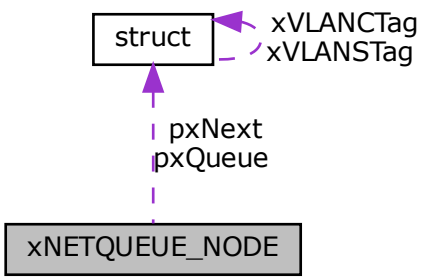
- source/include/FreeRTOS_TSN_NetworkSchedulerQueue.h

4.10 xNETQUEUE_NODE Struct Reference

This is the structure the stores the nodes of the network scheduler.

```
#include <FreeRTOS_TSN_NetworkSchedulerBlock.h>
```

Collaboration diagram for xNETQUEUE_NODE:



Data Fields

- uint8_t ucNumChildren
- void * pvScheduler
- struct xNETQUEUE * pxQueue
- struct xNETQUEUE_NODE * pxNext [ ]

4.10.1 Detailed Description

This is the structure the stores the nodes of the network scheduler.

A generic node has either other nodes as children, specified in the pxNext array which has size ucNumChildren, of a pxQueue, which is a leaf in the network scheduler, and in that case ucNumChildren and pxNext will be ignored. The two cases are distinguished by checking if pxQueue is NULL.

4.10.2 Field Documentation

4.10.2.1 pvScheduler

```
void* xNETQUEUE_NODE::pvScheduler
```

Pointer to the scheduler which stores the ready and select function pointers

4.10.2.2 pxNext

```
struct xNETQUEUE_NODE* xNETQUEUE_NODE::pxNext[]
```

The array which stores pointer to the children of this node

4.10.2.3 pxQueue

```
struct xNETQUEUE* xNETQUEUE_NODE::pxQueue
```

Pointer to a leaf of the scheduler. If this NULL, than the scheduler will recurse in the children stored in the pxNext field

4.10.2.4 ucNumChildren

```
uint8_t xNETQUEUE_NODE::ucNumChildren
```

Number of children nodes in pxNext array. If pxQueue is not NULL this is ignored

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_NetworkSchedulerBlock.h

4.11 xNETWORK_INTERFACE_CONFIG Struct Reference

`#include <NetworkWrapper.h>`

Data Fields

- BaseType_t xEMACIndex
- BaseType_t xNumTags
- uint16_t usVLANTag
- uint16_t usServiceVLANTag

4.11.1 Field Documentation

4.11.1.1 usServiceVLANTag

`uint16_t xNETWORK_INTERFACE_CONFIG::usServiceVLANTag`

4.11.1.2 usVLANTag

`uint16_t xNETWORK_INTERFACE_CONFIG::usVLANTag`

4.11.1.3 xEMACIndex

`BaseType_t xNETWORK_INTERFACE_CONFIG::xEMACIndex`

4.11.1.4 xNumTags

`BaseType_t xNETWORK_INTERFACE_CONFIG::xNumTags`

The documentation for this struct was generated from the following file:

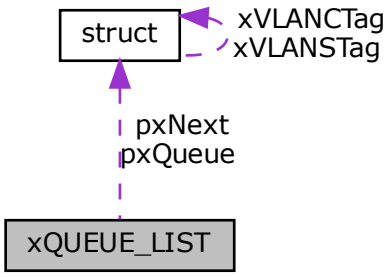
- source/portable/NetworkInterface/include/NetworkWrapper.h

4.12 xQUEUE_LIST Struct Reference

A list of network queue pointer.

`#include <FreeRTOS_TSN_NetworkScheduler.h>`

Collaboration diagram for xQUEUE_LIST:



Data Fields

- struct xNETQUEUE * pxQueue
- struct xQUEUE_LIST * pxNext

4.12.1 Detailed Description

A list of network queue pointer.

This will keep all the queue in a list structure and will help lookup a specific queue without the need to traverse the tree structure.

4.12.2 Field Documentation

4.12.2.1 pxNext

```
struct xQUEUE_LIST* xQUEUE_LIST::pxNext
```

4.12.2.2 pxQueue

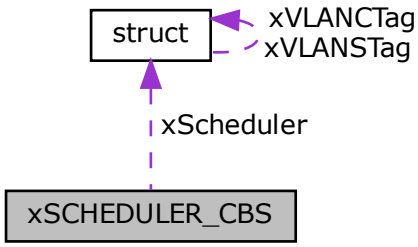
```
struct xNETQUEUE* xQUEUE_LIST::pxQueue
```

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_NetworkScheduler.h

4.13 xSCHEDULER_CBS Struct Reference

Collaboration diagram for xSCHEDULER_CBS:



Data Fields

- struct xSCHEDULER_GENERIC xScheduler
- UBaseType_t uxBandwidth
- UBaseType_t uxMaxCredit
- TickType_t uxNextActivation

4.13.1 Field Documentation

4.13.1.1 uxBandwidth

```
UBaseType_t xSCHEDULER_CBS::uxBandwidth
```

4.13.1.2 uxMaxCredit

```
UBaseType_t xSCHEDULER_CBS::uxMaxCredit
```

4.13.1.3 uxNextActivation

```
TickType_t xSCHEDULER_CBS::uxNextActivation
```

4.13.1.4 xScheduler

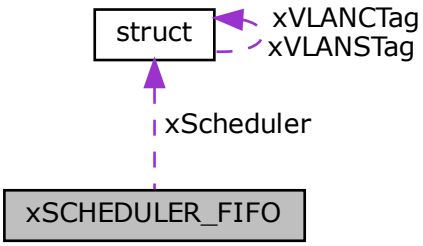
```
struct xSCHEDULER_GENERIC xSCHEDULER_CBS::xScheduler
```

The documentation for this struct was generated from the following file:

- source/modules/CreditBasedScheduler/SchedCBS.c

4.14 xSCHEDULER_FIFO Struct Reference

Collaboration diagram for xSCHEDULER_FIFO:



Data Fields

- struct xSCHEDULER_GENERIC xScheduler

4.14.1 Field Documentation

4.14.1.1 xScheduler

```
struct xSCHEDULER_GENERIC xSCHEDULER_FIFO::xScheduler
```

The documentation for this struct was generated from the following file:

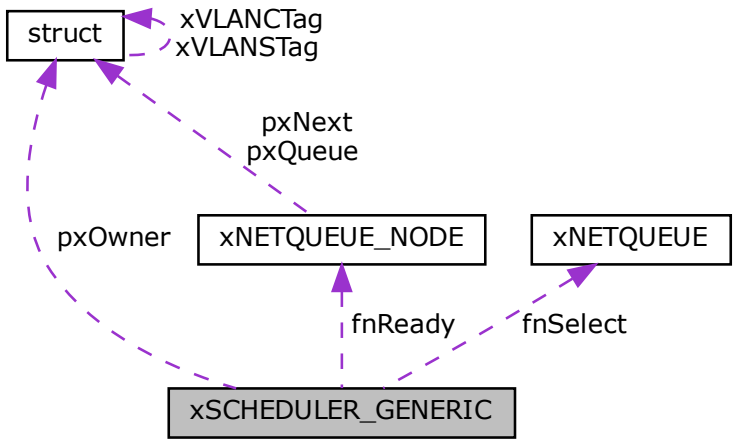
- source/modules/BasicSchedulers/BasicSchedulers.c

4.15 xSCHEDULER_GENERIC Struct Reference

A generic structure for implementing a scheduler.

`#include <FreeRTOS_TSN_NetworkSchedulerBlock.h>`

Collaboration diagram for xSCHEDULER_GENERIC:



Data Fields

- uint16_t usSize
- struct xNETQUEUE_NODE * pxOwner
- SelectQueueFunction_t fnSelect
- ReadyQueueFunction_t fnReady
- char ucAttributes [ ]

4.15.1 Detailed Description

A generic structure for implementing a scheduler.

This is not indented to be used as is, but should be implemented inside a network scheduler as first member. The size is the length of the specilized version of this struct, which also takes into consideration data iin ucAttributes. The select function should return a pointer to chosen network queue in the entire subtree spanned by the owner, or NULL if not queue can be scheduled. The ready function should return pdTRUE if the queue is allowed to schedule a packet, or pdFALSE if not.

4.15.2 Field Documentation

4.15.2.1 fnReady

```
ReadyQueueFunction_t xSCHEDULER_GENERIC::fnReady
```

Function to determine if the underlining network node is allowed to schedule a packet

4.15.2.2 fnSelect

```
SelectQueueFunction_t xSCHEDULER_GENERIC::fnSelect
```

Function to select a children of the owner network node

4.15.2.3 pxOwner

```
struct xNETQUEUE_NODE* xSCHEDULER_GENERIC::pxOwner
```

Pointer to the node in which this scheduler is used

4.15.2.4 ucAttributes

```
char xSCHEDULER_GENERIC::ucAttributes[]
```

This contains the attributes of the different scheduler implementations

4.15.2.5 usSize

```
uint16_t xSCHEDULER_GENERIC::usSize
```

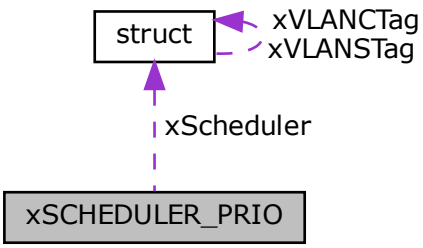
The total length of this structure, counting also the flexible members

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_NetworkSchedulerBlock.h

4.16 xSCHEDULER_PRIO Struct Reference

Collaboration diagram for xSCHEDULER_PRIO:



Data Fields

- struct xSCHEDULER_GENERIC xScheduler

4.16.1 Field Documentation

4.16.1.1 xScheduler

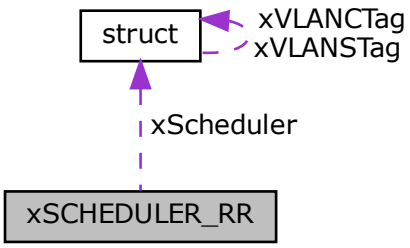
```
struct xSCHEDULER_GENERIC xSCHEDULER_PRIO::xScheduler
```

The documentation for this struct was generated from the following file:

- source/modules/BasicSchedulers/BasicSchedulers.c

4.17 xSCHEDULER_RR Struct Reference

Collaboration diagram for xSCHEDULER_RR:



Data Fields

- struct xSCHEDULER_GENERIC xScheduler

4.17.1 Field Documentation

4.17.1.1 xScheduler

```
struct xSCHEDULER_GENERIC xSCHEDULER_RR::xScheduler
```

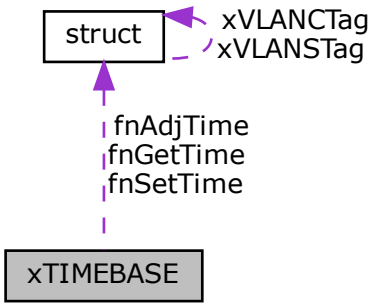
The documentation for this struct was generated from the following file:

- source/modules/BasicSchedulers/BasicSchedulers.c

4.18 xTIMEBASE Struct Reference

```
#include <FreeRTOS_TSN_Timebase.h>
```

Collaboration diagram for xTIMEBASE:



Data Fields

- TimeBaseStartFunction_t fnStart
- TimeBaseStopFunction_t fnStop
- TimeBaseSetTimeFunction_t fnSetTime
- TimeBaseGetTimeFunction_t fnGetTime
- TimeBaseAdjTimeFunction_t fnAdjTime

4.18.1 Field Documentation

4.18.1.1 fnAdjTime

```
TimeBaseAdjTimeFunction_t xTIMEBASE::fnAdjTime
```

4.18.1.2 fnGetTime

`TimeBaseGetTimeFunction_t xTIMEBASE::fnGetTime`

4.18.1.3 fnSetTime

`TimeBaseSetTimeFunction_t xTIMEBASE::fnSetTime`

4.18.1.4 fnStart

`TimeBaseStartFunction_t xTIMEBASE::fnStart`

4.18.1.5 fnStop

`TimeBaseStopFunction_t xTIMEBASE::fnStop`

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_Timebase.h

4.19 xTSN_SOCKET Struct Reference

`#include <FreeRTOS_TSN_Sockets.h>`

Data Fields

- Socket_t xBaseSocket
- QueueHandle_t xErrQueue
- uint8_t ucDSClass
- uint32_t ulTSFlags
- ListItem_t xBoundSocketListItem
- TaskHandle_t xSendTask
- TaskHandle_t xRecvTask

4.19.1 Field Documentation

4.19.1.1 ucDSClass

```
uint8_t xTSN_SOCKET::ucDSClass
```

Differentiated services class

4.19.1.2 ulTSFlags

```
uint32_t xTSN_SOCKET::ulTSFlags
```

Holds the timestamping config bits

4.19.1.3 xBaseSocket

```
Socket_t xTSN_SOCKET::xBaseSocket
```

Reuse the same socket structure as Plus-TCP addon

4.19.1.4 xBoundSocketListItem

```
ListItem_t xTSN_SOCKET::xBoundSocketListItem
```

4.19.1.5 xErrQueue

```
QueueHandle_t xTSN_SOCKET::xErrQueue
```

Contain errqueue with ancillary msgs

4.19.1.6 xRecvTask

```
TaskHandle_t xTSN_SOCKET::xRecvTask
```

Task handle of the task who is receiving (should always be at most one)

4.19.1.7 xSendTask

```
TaskHandle_t xTSN_SOCKET::xSendTask
```

To keep track of TSN sockets Task handle of the task who is sending (should always be at most one)

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_Sockets.h

4.20 xVLAN_TAG Struct Reference

`#include <FreeRTOS_TSN_VLANTags.h>`

Data Fields

- uint16_t usTPID
- uint16_t usTCI

4.20.1 Field Documentation

4.20.1.1 usTCI

`uint16_t xVLAN_TAG::usTCI`

4.20.1.2 usTPID

`uint16_t xVLAN_TAG::usTPID`

The documentation for this struct was generated from the following file:

- source/include/FreeRTOS_TSN_VLANTags.h

Chapter 5

File Documentation

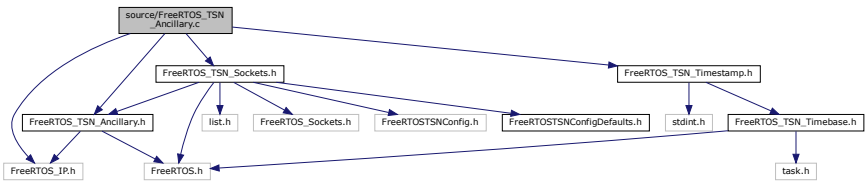
5.1 README.md File Reference

5.2 source/FreeRTOS_TSN_Ancillary.c File Reference

Implementation of ancillary message functions for FreeRTOS+TCP.

```
#include "FreeRTOS_TSN_Ancillary.h"
#include "FreeRTOS_IP.h"
#include "FreeRTOS_TSN_Sockets.h"
#include "FreeRTOS_TSN_Timestamp.h"
```

Include dependency graph for FreeRTOS_TSN_Ancillary.c:



Functions

- portINLINE struct cmsghdr * __CMSG_NXTHDR (void *ctl, size_t size, struct cmsghdr *cmsg)

  Aligns the size of a control message buffer.
- struct msghdr * pxAncillaryMsgMalloc ()

  Allocates memory for a new msghdr structure.
- void vAncillaryMsgFree (struct msghdr *pxMsgh)

  Frees the memory allocated for an ancillary message.
- void vAncillaryMsgFreeAll (struct msghdr *pxMsgh)

  Frees a msghdr.
- BaseType_t xAncillaryMsgFillName (struct msghdr *pxMsgh, IP_Address_t *xAddr, uint16_t usPort, BaseType_t xFamily)

  Fills in the name field of a message header structure with the given IP address, port, and family.

- void vAncillaryMsgFreeName (struct msghdr *pxMsgh)

  Frees the memory allocated for the name field in the given msghdr structure.
- BaseType_t xAncillaryMsgFillPayload (struct msghdr *pxMsgh, uint8_t *pucBuffer, size_t uxLength)

  Fills the payload of an ancillary message.
- void vAncillaryMsgFreePayload (struct msghdr *pxMsgh)

  Frees the payload of an ancillary message.
- BaseType_t xAncillaryMsgControlFill (struct msghdr *pxMsgh, struct cmsghdr *pxCmsgVec, void **ppv↩ DataVec, size_t *puxDataLenVec, size_t uxNumBuffers)

  Fills the ancillary message control structure with data.
- BaseType_t xAncillaryMsgControlFillSingle (struct msghdr *pxMsgh, struct cmsghdr *pxCmsg, void *pvData, size_t puxDataLen)

  Fills a single ancillary message control structure with data.
- void vAncillaryMsgFreeControl (struct msghdr *pxMsgh)

  Frees the memory allocated for the ancillary message control data.

5.2.1 Detailed Description

Implementation of ancillary message functions for FreeRTOS+TCP.

This file contains the implementation of ancillary message functions for FreeRTOS+TCP. These functions are used to allocate, fill, and free ancillary messages, which are used to pass control and data information between sockets.

5.2.2 Function Documentation

5.2.2.1 __CMSG_NXTHDR()

```
portINLINE struct cmsghdr * __CMSG_NXTHDR (
             void * ctl,
             size_t size,
             struct cmsghdr * cmsg )
```

Aligns the size of a control message buffer.

5.2.2.2 pxAncillaryMsgMalloc()

```
struct msghdr * pxAncillaryMsgMalloc ( )
```

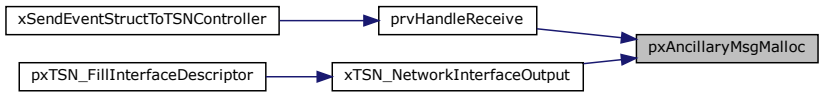
Allocates memory for a new msghdr structure.

This function allocates memory for a new msghdr structure using the pvPortMalloc function. The allocated memory is then initialized with zeros using the memset function.

Returns

A pointer to the newly allocated msghdr structure.

Here is the caller graph for this function:



5.2.2.3 vAncillaryMsgFree()

```
void vAncillaryMsgFree (
            struct msghdr * pxMsgh )
```

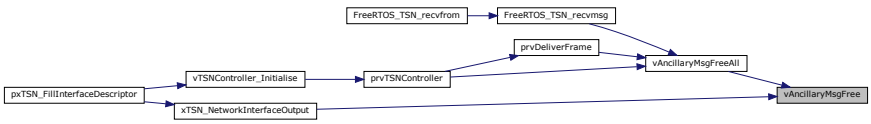
Frees the memory allocated for an ancillary message.

This function frees the memory allocated for the given ancillary message.

Parameters

pxMsgh	Pointer to the `msghdr` structure representing the ancillary message.

Here is the caller graph for this function:



5.2.2.4 vAncillaryMsgFreeAll()

```
void vAncillaryMsgFreeAll (
            struct msghdr * pxMsgh )
```

Frees a msghdr.

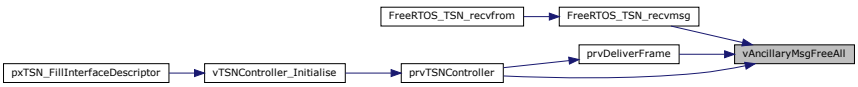
This will free all the non null members of the msghdr. In order to make sense it should always be used on a msghdr created using pxAncillaryMsgMalloc(), which takes the duty of initializing the struct to zero. Also note that this frees the iovec array, but not the iov_base buffers.

Parameters

pxMsgh	Pointer to msghdr to free

Here is the call graph for this function:

Here is the caller graph for this function:



5.2.2.5 vAncillaryMsgFreeControl()

```
void vAncillaryMsgFreeControl (
            struct msghdr * pxMsgh )
```

Frees the memory allocated for the ancillary message control data.

This function frees the memory allocated for the ancillary message control data pointed to by the `msg_control` member of the `msghdr` structure.

Parameters

pxMsgh	Pointer to the `msghdr` structure.

5.2.2.6 vAncillaryMsgFreeName()

```
void vAncillaryMsgFreeName (
            struct msghdr * pxMsgh )
```

Frees the memory allocated for the name field in the given msghdr structure.

This function frees the memory allocated for the name field in the provided msghdr structure.

Parameters

pxMsgh	Pointer to the msghdr structure.

5.2.2.7 vAncillaryMsgFreePayload()

```
void vAncillaryMsgFreePayload (
            struct msghdr * pxMsgh )
```

Frees the payload of an ancillary message.

This function frees the memory allocated for the payload of an ancillary message. The caller must ensure that the array is not empty before calling this function.

Parameters

pxMsgh	Pointer to the msghdr structure representing the ancillary message.

5.2.2.8 xAncillaryMsgControlFill()

```
BaseType_t xAncillaryMsgControlFill (
            struct msghdr * pxMsgh,
            struct cmsghdr * pxCmsgVec,
            void ** ppvDataVec,
            size_t * puxDataLenVec,
            size_t uxNumBuffers )
```

Fills the ancillary message control structure with data.

This function fills the ancillary message control structure with data provided in the input parameters.

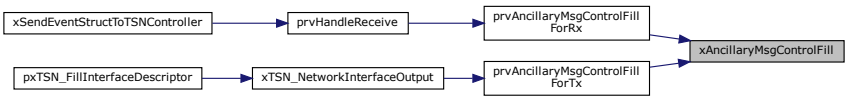
Parameters

pxMsgh	A pointer to the msghdr structure representing the message header.
pxCmsgVec	A pointer to the cmsghdr structure representing the control message vector.
ppvDataVec	An array of void pointers representing the data vector.
puxDataLenVec	An array of size_t values representing the data length vector.
uxNumBuffers	The number of buffers in the data vector.

Returns

pdTRUE if the ancillary message control structure is successfully filled, pdFAIL otherwise.

Here is the caller graph for this function:



5.2.2.9 xAncillaryMsgControlFillSingle()

```
BaseType_t xAncillaryMsgControlFillSingle (
            struct msghdr * pxMsgh,
            struct cmsghdr * pxCmsg,
            void * pvData,
            size_t puxDataLen )
```

Fills a single ancillary message control structure with data.

This function fills a single ancillary message control structure with data.

Parameters

pxMsgh	Pointer to the msghdr structure.
pxCmsg	Pointer to the cmsghdr structure.
pvData	Pointer to the data.
puxDataLen	Length of the data.

Returns

The result of the operation.

5.2.2.10 xAncillaryMsgFillName()

```
BaseType_t xAncillaryMsgFillName (
            struct msghdr * pxMsgh,
            IP_Address_t * xAddr,
            uint16_t usPort,
            BaseType_t xFamily )
```

Fills in the name field of a message header structure with the given IP address, port, and family.

This function is used to fill in the name field of a message header structure with the given IP address, port, and family. The name field is used to specify the source or destination address of a socket.

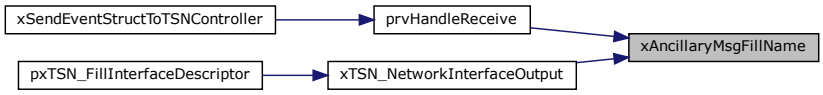
Parameters

pxMsgh	A pointer to the message header structure.
xAddr	A pointer to the IP address to be filled in the name field.
usPort	The port number to be filled in the name field.
xFamily	The address family to be filled in the name field.

Returns

pdPASS if the name field is successfully filled, pdFAIL otherwise.

Here is the caller graph for this function:



5.2.2.11 xAncillaryMsgFillPayload()

```
BaseType_t xAncillaryMsgFillPayload (
            struct msghdr * pxMsgh,
            uint8_t * pucBuffer,
            size_t uxLength )
```

Fills the payload of an ancillary message.

This function fills the payload of an ancillary message with the provided buffer and length.

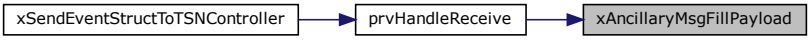
Parameters

pxMsgh	Pointer to the msghdr structure representing the ancillary message.
pucBuffer	Pointer to the buffer containing the payload data.
uxLength	Length of the payload data in bytes.

Returns

pdPASS if the payload was successfully filled, pdFAIL otherwise.

Here is the caller graph for this function:

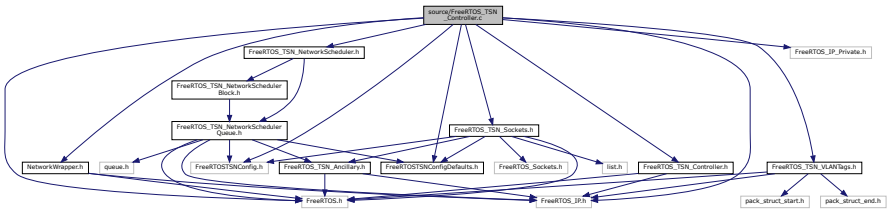


5.3 source/FreeRTOS_TSN_Controller.c File Reference

FreeRTOS TSN Controller implementation.

```
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"
#include "FreeRTOS_IP_Private.h"
#include "FreeRTOSTSNConfig.h"
#include "FreeRTOSTSNConfigDefaults.h"
#include "FreeRTOS_TSN_Controller.h"
#include "FreeRTOS_TSN_NetworkScheduler.h"
#include "FreeRTOS_TSN_VLANTags.h"
#include "FreeRTOS_TSN_Sockets.h"
#include "NetworkWrapper.h"
```

Include dependency graph for FreeRTOS_TSN_Controller.c:



Macros

- #define controllerTSN_TASK_BASE_PRIO (tsnconfigTSN_CONTROLLER_PRIORITY)

Functions

- void prvReceiveUDPPacketTSN (NetworkQueueItem_t *pxItem, TSNSocket_t xTSNSocket, Socket_t x↩ BaseSocket)

  Receives a UDP packet for a TSN socket.
- void prvDeliverFrame (NetworkQueueItem_t *pxItem, BaseType_t xUsingIPTask)

  Function to deliver a network frame to the appropriate socket.
- static void prvTSNController (void *pvParameters)

  TSN Controller task function.
- void vTSNController_Initialise (void)

  Function to initialize the TSN Controller task.
- BaseType_t xNotifyController ()

  Function to notify the TSN Controller task.
- void vTSNControllerComputePriority (void)

  Function to compute the priority of the TSN Controller task.
- BaseType_t xTSNControllerUpdatePriority (UBaseType_t uxPriority)

  Function to update the priority of the TSN Controller task.
- BaseType_t xIsCallingFromTSNController (void)

  Function to check if the caller task is the TSN Controller task.

Variables

- static TaskHandle_t xTSNControllerHandle = NULL
- NetworkQueueList_t * pxNetworkQueueList

5.3.1 Detailed Description

FreeRTOS TSN Controller implementation.

This file contains the implementation of the FreeRTOS TSN Controller, which is responsible for handling incoming network packets and forwarding them to the appropriate tasks or the IP task.

5.3.2 Macro Definition Documentation

5.3.2.1 controllerTSN_TASK_BASE_PRIO

```
#define controllerTSN_TASK_BASE_PRIO ( tsnconfigTSN_CONTROLLER_PRIORITY )
```

5.3.3 Function Documentation

5.3.3.1 prvDeliverFrame()

```
void prvDeliverFrame (
            NetworkQueueItem_t * pxItem,
            BaseType_t xUsingIPTask )
```

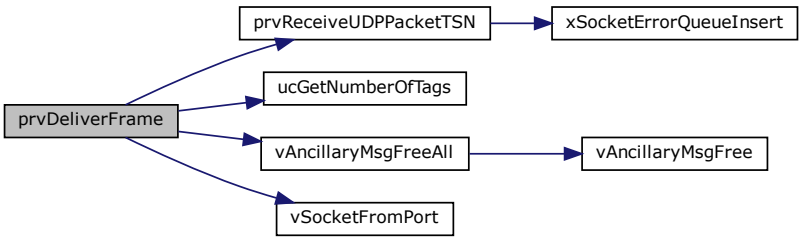
Function to deliver a network frame to the appropriate socket.

This function is responsible for delivering a network frame to the appropriate socket based on the frame type.

Parameters

in	*pxItem*	Pointer to the network buffer descriptor

Here is the call graph for this function:



Here is the caller graph for this function:

5.3.3.2 prvReceiveUDPPacketTSN()

```
void prvReceiveUDPPacketTSN (
            NetworkQueueItem_t * pxItem,
            TSNSocket_t xTSNSocket,
            Socket_t xBaseSocket )
```

Receives a UDP packet for a TSN socket.

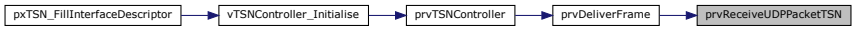
This function is responsible for receiving a UDP packet for a TSN socket. It handles error conditions and inserts the packet into the waiting packets list. It also sets the receive event for the socket and updates the select group and user semaphore if applicable.

Parameters

pxItem	Pointer to the NetworkQueueItem_t structure containing the received packet.
xTSNSocket	The TSN socket to receive the packet for.
xBaseSocket	The base socket associated with the TSN socket.

Here is the call graph for this function:

Here is the caller graph for this function:



5.3.3.3 prvTSNController()

```
static void prvTSNController (
            void * pvParameters )  [static]
```

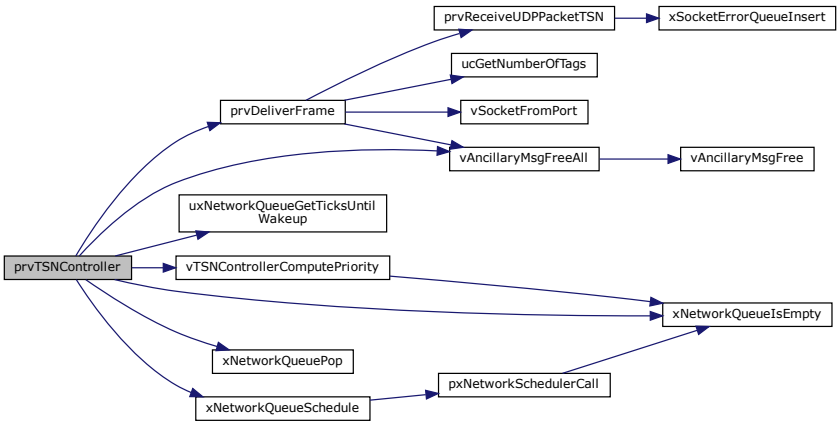
TSN Controller task function.

This function is the entry point for the TSN Controller task. It waits for notifications and processes network packets or events based on the notification received.

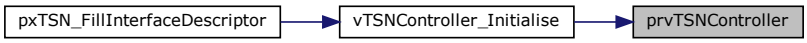
Parameters

in	*pvParameters*	Pointer to the task parameters (not used)

Here is the call graph for this function:



Here is the caller graph for this function:

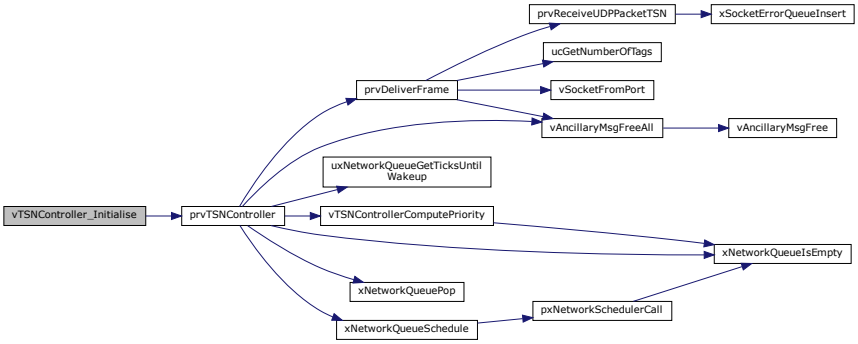


5.3.3.4 vTSNController_Initialise()

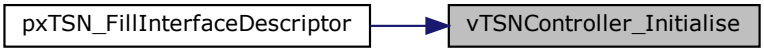
```
void vTSNController_Initialise (
            void  )
```

Function to initialize the TSN Controller task.

This function creates the TSN Controller task and sets its priority. Here is the call graph for this function:



Here is the caller graph for this function:

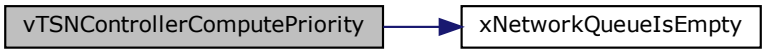


5.3.3.5 vTSNControllerComputePriority()

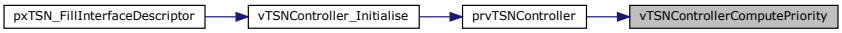
```
void vTSNControllerComputePriority (
            void  )
```

Function to compute the priority of the TSN Controller task.

The priority of the TSN controller is the maximum IPV among all the queues which has pending messages. Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.6 xIsCallingFromTSNController()

```
BaseType_t xIsCallingFromTSNController (
            void  )
```

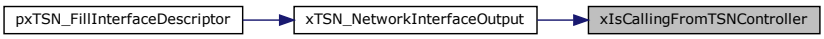
Function to check if the caller task is the TSN Controller task.

This function checks if the caller task is the TSN Controller task.

Returns

pdTRUE if the current task is the TSN Controller task, pdFALSE otherwise

Here is the caller graph for this function:



5.3.3.7 xNotifyController()

```
BaseType_t xNotifyController ( )
```

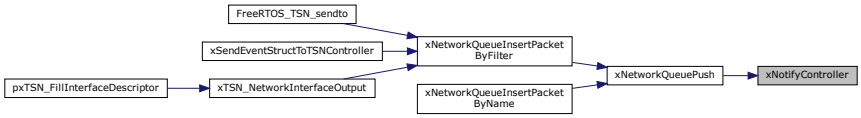
Function to notify the TSN Controller task.

This function notifies the TSN Controller task to wake up and process pending network packets or events.

Returns

pdTRUE if the notification is sent successfully, pdFALSE otherwise

Here is the caller graph for this function:



5.3.3.8 xTSNControllerUpdatePriority()

```
BaseType_t xTSNControllerUpdatePriority (
            UBaseType_t uxPriority )
```

Function to update the priority of the TSN Controller task.

This function updates the priority of the TSN Controller task if the new priority is higher than the current priority.

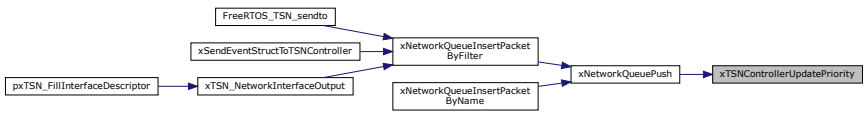
Parameters

in	*uxPriority*	New priority for the TSN Controller task

Returns

pdTRUE if the priority is updated, pdFALSE otherwise

Here is the caller graph for this function:



5.3.4 Variable Documentation

5.3.4.1 pxNetworkQueueList

```
NetworkQueueList_t* pxNetworkQueueList  [extern]
```

5.3.4.2 xTSNControllerHandle

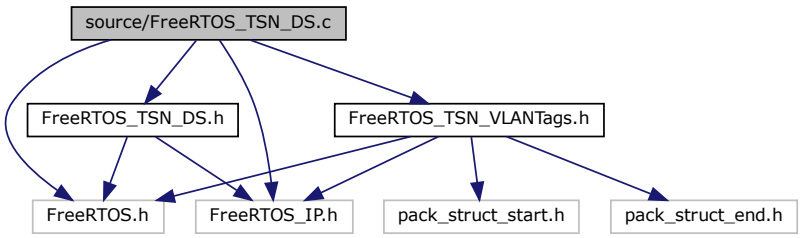
```
TaskHandle_t xTSNControllerHandle = NULL  [static]
```

5.4 source/FreeRTOS_TSN_DS.c File Reference

FreeRTOS TSN Compatibility Layer - Data Structures.

```
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"
#include "FreeRTOS_TSN_VLANTags.h"
#include "FreeRTOS_TSN_DS.h"
```

Include dependency graph for FreeRTOS_TSN_DS.c:



Functions

- void prvGetIPVersionAndOffset (NetworkBufferDescriptor_t *pxBuf, uint16_t *pusIPVersion, size_t *pulOffset)

  Retrieves the IP version and offset from the given network buffer.
- uint8_t ucDSClassGet (NetworkBufferDescriptor_t *pxBuf)

  Retrieves the DiffServ class from the given network buffer.
- BaseType_t xDSClassSet (NetworkBufferDescriptor_t *pxBuf, uint8_t ucValue)

  Sets the DiffServ class for the given network buffer.

5.4.1 Detailed Description

FreeRTOS TSN Compatibility Layer - Data Structures.

This file contains the implementation of functions related to retrieving and setting DiffServ class in network buffers.

5.4.2 Function Documentation

5.4.2.1 prvGetIPVersionAndOffset()

```
void prvGetIPVersionAndOffset (
            NetworkBufferDescriptor_t * pxBuf,
            uint16_t * pusIPVersion,
            size_t * pulOffset )
```

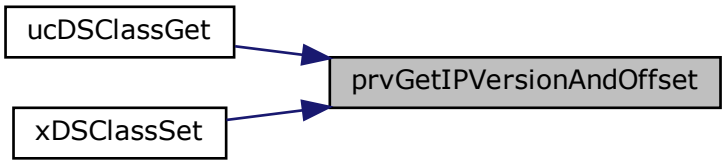
Retrieves the IP version and offset from the given network buffer.

This function extracts the IP version and offset from the Ethernet header of the network buffer.

Parameters

in	*pxBuf*	The network buffer descriptor.
out	*pusIPVersion*	Pointer to store the IP version.
out	*pulOffset*	Pointer to store the offset.

Here is the caller graph for this function:



5.4.2.2 ucDSClassGet()

```
uint8_t ucDSClassGet (
            NetworkBufferDescriptor_t * pxBuf )
```

Retrieves the DiffServ class from the given network buffer.

This function extracts the DiffServ class from the IP header of the network buffer based on the IP version.

Parameters

in	*pxBuf*	The network buffer descriptor.

Returns

The DiffServ class value.

Here is the call graph for this function:

5.4.2.3 xDSClassSet()

```
BaseType_t xDSClassSet (
            NetworkBufferDescriptor_t * pxBuf,
            uint8_t ucValue )
```

Sets the DiffServ class for the given network buffer.

This function sets the DiffServ class in the IP header of the network buffer based on the IP version.

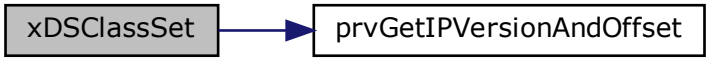
Parameters

in	*pxBuf*	The network buffer descriptor.
in	*ucValue*	The DiffServ class value to set.

Returns

pdPASS if the DiffServ class was set successfully, pdFAIL otherwise.

Here is the call graph for this function:

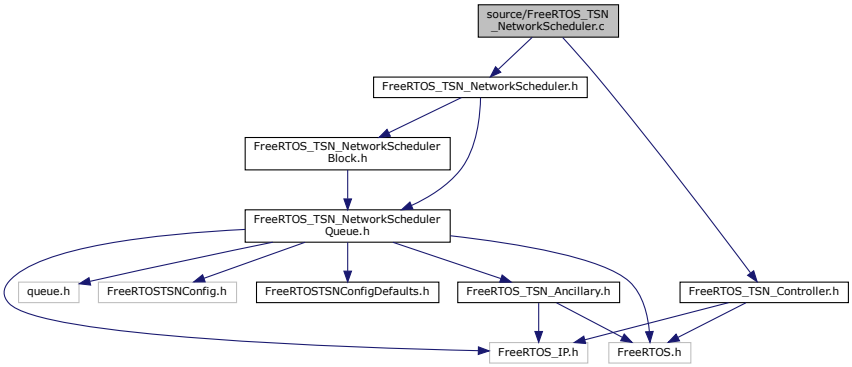


5.5 source/FreeRTOS_TSN_NetworkScheduler.c File Reference

This file contains the implementation of the network scheduler for FreeRTOS TSN Compatibility Layer.

```
#include "FreeRTOS_TSN_NetworkScheduler.h"
#include "FreeRTOS_TSN_Controller.h"
```

Include dependency graph for FreeRTOS_TSN_NetworkScheduler.c:



Functions

- BaseType_t prvMatchQueuePolicy (const NetworkQueueItem_t *pxItem, NetworkQueue_t *pxQueue)

  Matches the filtering policy of a network queue with a network queue item.
- void vNetworkQueueListAdd (NetworkQueueList_t *pxItem)

  Adds a network queue to the network queue list.
- BaseType_t xNetworkQueueAssignRoot (NetworkNode_t *pxNode)

  Assigns the root network node.
- BaseType_t xNetworkQueueInsertPacketByFilter (const NetworkQueueItem_t *pxItem, UBaseType_t ux↩ Timeout)

  Iterate over the list of network queues and find a match based on the queues' filtering policy. If more than one queue matches the filter, the one with the highest IPV (Internet Protocol Version) is chosen.
- BaseType_t xNetworkQueueInsertPacketByName (const NetworkQueueItem_t *pxItem, char *pcQueue↩ Name, UBaseType_t uxTimeout)

  Inserts a network queue item into a network queue based on the queue name.
- NetworkQueue_t * xNetworkQueueSchedule (void)

  Schedules the network queues and returns the chosen network queue.
- BaseType_t xNetworkQueuePush (NetworkQueue_t *pxQueue, const NetworkQueueItem_t *pxItem, UBaseType_t uxTimeout)

  Pushes a network queue item into a network queue.
- BaseType_t xNetworkQueuePop (NetworkQueue_t *pxQueue, NetworkQueueItem_t *pxItem, UBaseType↩ _t uxTimeout)

  Pops a network queue item from a network queue.

Variables

- NetworkNode_t * pxNetworkQueueRoot = NULL
- NetworkQueueList_t * pxNetworkQueueList = NULL
- UBaseType_t uxNumQueues = 0

5.5.1 Detailed Description

This file contains the implementation of the network scheduler for FreeRTOS TSN Compatibility Layer.

5.5.2 Function Documentation

5.5.2.1 prvMatchQueuePolicy()

```
BaseType_t prvMatchQueuePolicy (
            const NetworkQueueItem_t * pxItem,
            NetworkQueue_t * pxQueue )
```

Matches the filtering policy of a network queue with a network queue item.

This function compares the filtering policy of a network queue with a network queue item to determine if they match. The matching is based on the event type of the network queue item.

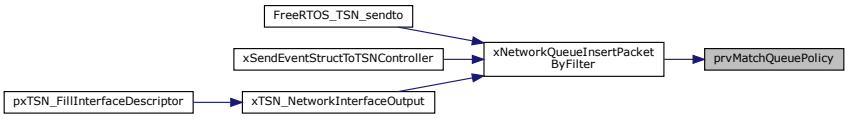
Parameters

pxItem	The network queue item to match.
pxQueue	The network queue to match against.

Returns

pdTRUE if the network queue item matches the filtering policy of the network queue, pdFALSE otherwise.

Here is the caller graph for this function:



5.5.2.2 vNetworkQueueListAdd()

```
void vNetworkQueueListAdd (
            NetworkQueueList_t * pxItem )
```

Adds a network queue to the network queue list.

This function is used to add a network queue to the network queue list. The network queue list is a linked list that keeps track of all the network queues.

Parameters

pxItem	The network queue to add.

5.5.2.3 xNetworkQueueAssignRoot()

```
BaseType_t xNetworkQueueAssignRoot (
            NetworkNode_t * pxNode )
```

Assigns the root network node.

This function assigns the specified network node as the root node for the TSN controller's scheduling function. The root node is the starting point for the scheduling algorithm.

Parameters

pxNode	The network node to assign as the root.

Returns

pdPASS if the root network node is successfully assigned, pdFAIL otherwise.

5.5.2.4 xNetworkQueueInsertPacketByFilter()

```
BaseType_t xNetworkQueueInsertPacketByFilter (
            const NetworkQueueItem_t * pxItem,
            UBaseType_t uxTimeout )
```

Iterate over the list of network queues and find a match based on the queues' filtering policy. If more than one queue matches the filter, the one with the highest IPV (Internet Protocol Version) is chosen.

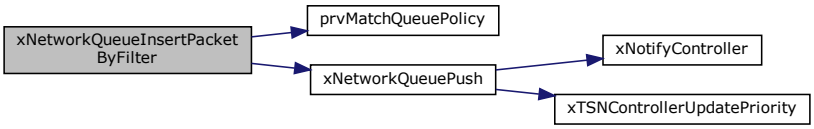
Parameters

pxItem	The network queue item to insert.
uxTimeout	The timeout value for the insertion operation.

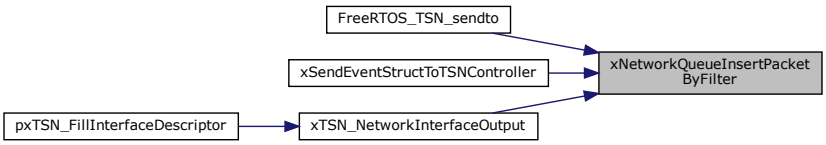
Returns

pdPASS if the network queue item is successfully inserted, pdFAIL otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.5 xNetworkQueueInsertPacketByName()

```
BaseType_t xNetworkQueueInsertPacketByName (
            const NetworkQueueItem_t * pxItem,
            char * pcQueueName,
            UBaseType_t uxTimeout )
```

Inserts a network queue item into a network queue based on the queue name.

This function inserts a network queue item into a network queue identified by its name. The network queue item is inserted using the xNetworkQueuePush() function.

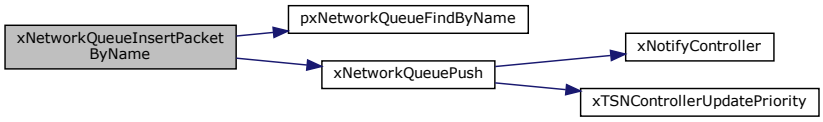
Parameters

pxItem	The network queue item to insert.
pcQueueName	The name of the network queue to insert into.
uxTimeout	The timeout value for the insertion operation.

Returns

pdPASS if the network queue item is successfully inserted, pdFAIL otherwise.

Here is the call graph for this function:



5.5.2.6 xNetworkQueuePop()

```
BaseType_t xNetworkQueuePop (
            NetworkQueue_t * pxQueue,
            NetworkQueueItem_t * pxItem,
            UBaseType_t uxTimeout )
```

Pops a network queue item from a network queue.

This function pops an item from the specified network queue. If the queue is empty, the function will wait for a specified timeout period for an item to become available.

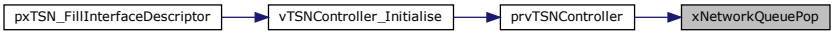
Parameters

pxQueue	The network queue to pop the item from.
pxItem	The network queue item to pop.
uxTimeout	The timeout value for the pop operation.

Returns

pdPASS if a network queue item is successfully popped, pdFAIL otherwise.

Here is the caller graph for this function:



5.5.2.7 xNetworkQueuePush()

```
BaseType_t xNetworkQueuePush (
             NetworkQueue_t * pxQueue,
             const NetworkQueueItem_t * pxItem,
             UBaseType_t uxTimeout )
```

Pushes a network queue item into a network queue.

This function pushes a network queue item into a network queue. It first calls the `fnOnPush` callback function if queue event callbacks are enabled. Then, it uses the `xQueueSendToBack` function to send the item to the back of the queue. If the item is successfully pushed, it updates the priority of the TSN controller (if dynamic priority is enabled), notifies the controller, and returns `pdPASS`. Otherwise, it returns `pdFAIL`.

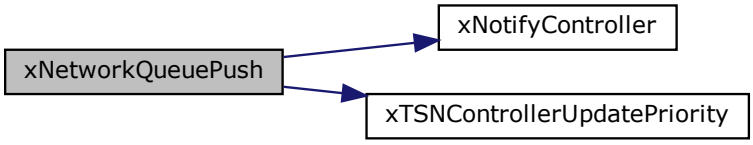
Parameters

pxQueue	The network queue to push the item into.
pxItem	The network queue item to push.
uxTimeout	The timeout value for the push operation.

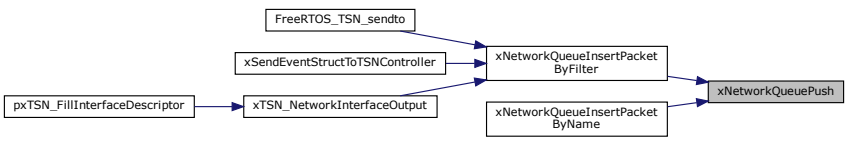
Returns

pdPASS if the network queue item is successfully pushed, pdFAIL otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.8 xNetworkQueueSchedule()

```
NetworkQueue_t * xNetworkQueueSchedule (
            void  )
```

Schedules the network queues and returns the chosen network queue.

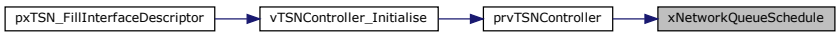
This function is responsible for scheduling the network queues and selecting the next network queue to be processed. It checks if there is a network queue available and calls the network scheduler function to make the selection. If there is no network queue available, it returns pdFAIL.

Returns

The chosen network queue, or pdFAIL if no network queue is available.

Here is the call graph for this function:

Here is the caller graph for this function:



5.5.3 Variable Documentation

5.5.3.1 pxNetworkQueueList

```
NetworkQueueList_t* pxNetworkQueueList = NULL
```

5.5.3.2 pxNetworkQueueRoot

```
NetworkNode_t* pxNetworkQueueRoot = NULL
```

5.5.3.3 uxNumQueues

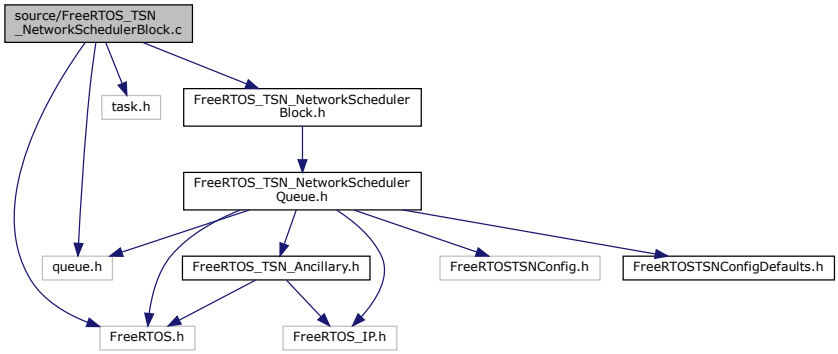
```
UBaseType_t uxNumQueues = 0
```

5.6 source/FreeRTOS_TSN_NetworkSchedulerBlock.c File Reference

Implementation of the FreeRTOS TSN Network Scheduler Block.

```
#include "FreeRTOS.h"
#include "queue.h"
#include "task.h"
#include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
```

Include dependency graph for FreeRTOS_TSN_NetworkSchedulerBlock.c:



Functions

- BaseType_t prvAlwaysReady (NetworkNode_t ∗pxNode)

  Default ready function for schedulers.
- NetworkQueue_t ∗ prvSelectFirst (NetworkNode_t ∗pxNode)

  Default ready function for schedulers.
- BaseType_t xNetworkSchedulerLinkQueue (NetworkNode_t ∗pxNode, NetworkQueue_t ∗pxQueue)

  Links a network queue to a network node.
- BaseType_t xNetworkSchedulerLinkChild (NetworkNode_t ∗pxNode, NetworkNode_t ∗pxChild, size_t ux↩ Position)

  Links a child network node to a parent network node.
- NetworkQueue_t ∗ pxNetworkSchedulerCall (NetworkNode_t ∗pxNode)

  Calls the network scheduler for a network node.
- NetworkBufferDescriptor_t ∗ pxPeekNextPacket (NetworkNode_t ∗pxNode)

  Peeks the next packet in a network node's queue.
- TickType_t uxNetworkQueueGetTicksUntilWakeup (void)

  Gets the ticks until the next wakeup event.
- void vNetworkQueueAddWakeupEvent (TickType_t uxTime)

  Adds a wakeup event to the network queue.

Variables

- TickType_t uxNextWakeup = 0

5.6.1 Detailed Description

Implementation of the FreeRTOS TSN Network Scheduler Block.

This file contains the implementation of the FreeRTOS TSN Network Scheduler Block. It provides functions for creating and releasing network nodes, linking queues and children to a node, selecting the first node, checking if a node is always ready, and calling the network scheduler. It also includes functions for peeking the next packet, getting the ticks until wakeup, and adding a wakeup event.

5.6.2 Function Documentation

5.6.2.1 prvAlwaysReady()

```
BaseType_t prvAlwaysReady (
            NetworkNode_t * pxNode )
```

Default ready function for schedulers.

Returns

Always return pdTRUE

5.6.2.2 prvSelectFirst()

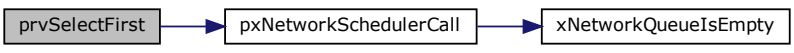
```
NetworkQueue_t * prvSelectFirst (
            NetworkNode_t * pxNode )
```

Default ready function for schedulers.

Returns

Always schedule first child

Here is the call graph for this function:



5.6.2.3 pxNetworkSchedulerCall()

```
NetworkQueue_t * pxNetworkSchedulerCall (
            NetworkNode_t * pxNode )
```

Calls the network scheduler for a network node.

This function is the core of the network scheduler. It will recursively call the ready function and the select function of the nodes, starting from the root until a ready node is found.

Parameters

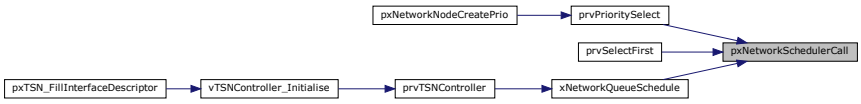
pxNode	Pointer to the network node.

Returns

Pointer to the selected network queue.

Here is the call graph for this function:

Here is the caller graph for this function:



5.6.2.4 pxPeekNextPacket()

```
NetworkBufferDescriptor_t * pxPeekNextPacket (
            NetworkNode_t * pxNode )
```

Peeks the next packet in a network node's queue.

This function is used to peek the next packet in a network node's queue.

Parameters

pxNode	Pointer to the network node.

Returns

Pointer to the next packet, or NULL if the queue is empty.

Here is the caller graph for this function:

5.6.2.5 uxNetworkQueueGetTicksUntilWakeup()

```
TickType_t uxNetworkQueueGetTicksUntilWakeup (
             void  )
```

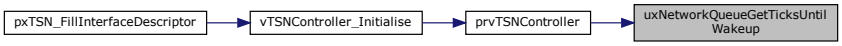
Gets the ticks until the next wakeup event.

This function is used to get the number of ticks until the next wakeup event.

Returns

Number of ticks until the next wakeup event.

Here is the caller graph for this function:



5.6.2.6 vNetworkQueueAddWakeupEvent()

```
void vNetworkQueueAddWakeupEvent (
             TickType_t uxTime )
```

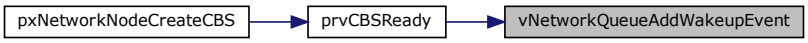
Adds a wakeup event to the network queue.

Altough the network scheduler will always periodicaly checks for new messages, calling this function can help speed up serving waiting packets. Any scheduler that has implemented a ready function that not always returns true should think of suggesting the TSN controller when to check again.

Parameters

uxTime	Time at which to add the wakeup event.

Here is the caller graph for this function:



5.6.2.7 xNetworkSchedulerLinkChild()

```
BaseType_t xNetworkSchedulerLinkChild (
            NetworkNode_t * pxNode,
            NetworkNode_t * pxChild,
            size_t uxPosition )
```

Links a child network node to a parent network node.

This function is used to link a child network node to a parent network node at the specified position.

Parameters

pxNode	Pointer to the parent network node.
pxChild	Pointer to the child network node.
uxPosition	Position at which to link the child network node.

Returns

pdPASS if the link is successful, pdFAIL otherwise.

5.6.2.8 xNetworkSchedulerLinkQueue()

```
BaseType_t xNetworkSchedulerLinkQueue (
            NetworkNode_t * pxNode,
            NetworkQueue_t * pxQueue )
```

Links a network queue to a network node.

This function is used to link a network queue to a network node.

Parameters

pxNode	Pointer to the network node.
pxQueue	Pointer to the network queue.

Returns

pdPASS if the link is successful, pdFAIL otherwise.

5.6.3 Variable Documentation

5.6.3.1 uxNextWakeup

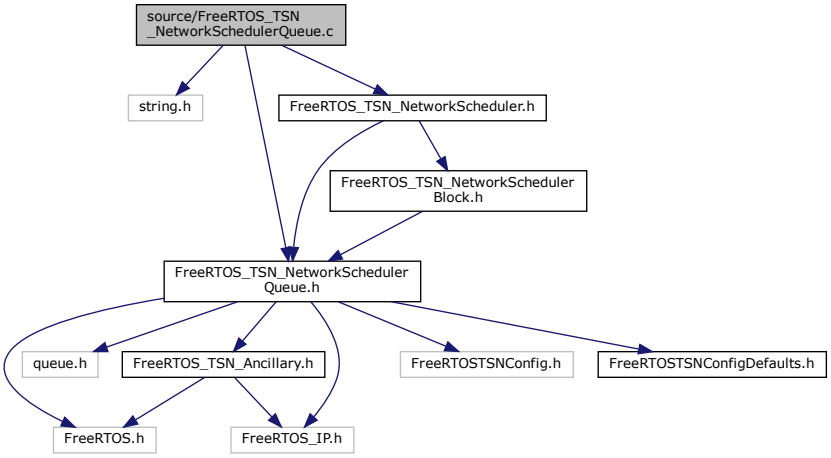
```
TickType_t uxNextWakeup = 0
```

5.7 source/FreeRTOS_TSN_NetworkSchedulerQueue.c File Reference

Implementation of the FreeRTOS TSN Network Scheduler Queue module.

```
#include <string.h>
#include "FreeRTOS_TSN_NetworkSchedulerQueue.h"
#include "FreeRTOS_TSN_NetworkScheduler.h"
```

Include dependency graph for FreeRTOS_TSN_NetworkSchedulerQueue.c:



Functions

- BaseType_t prvDefaultPacketHandler (NetworkBufferDescriptor_t ∗pxBuf)

  Default packet handler function.
- BaseType_t prvAlwaysTrue (NetworkBufferDescriptor_t ∗pxBuf)

  Function that always returns pdTRUE.
- UBaseType_t uxNetworkQueuePacketsWaiting (NetworkQueue_t ∗pxQueue)

  Get the number of packets waiting in a network queue.
- BaseType_t xNetworkQueueIsEmpty (NetworkQueue_t ∗pxQueue)

  Check if a network queue is empty.

5.7.1 Detailed Description

Implementation of the FreeRTOS TSN Network Scheduler Queue module.

This file contains the implementation of the Network Scheduler Queue module for the FreeRTOS TSN Compatibility Layer. It provides functions for creating, managing, and freeing network queues.

5.7.2 Function Documentation

5.7.2.1 prvAlwaysTrue()

```
BaseType_t prvAlwaysTrue (
            NetworkBufferDescriptor_t * pxBuf )
```

Function that always returns pdTRUE.

This function is used as a filter function that always returns pdTRUE. It is used when no filter function is provided for a network queue.

Parameters

pxBuf	The network buffer descriptor.

Returns

pdTRUE.

5.7.2.2 prvDefaultPacketHandler()

```
BaseType_t prvDefaultPacketHandler (
            NetworkBufferDescriptor_t * pxBuf )
```

Default packet handler function.

This function is used as the default packet handler for network buffers. It simply returns pdPASS without performing any action.

Parameters

pxBuf	The network buffer descriptor.

Returns

pdPASS.

5.7.2.3 uxNetworkQueuePacketsWaiting()

```
UBaseType_t uxNetworkQueuePacketsWaiting (
             NetworkQueue_t * pxQueue )
```

Get the number of packets waiting in a network queue.

This function returns the number of packets waiting in a network queue.

Parameters

pxQueue	A pointer to the network queue.

Returns

The number of packets waiting in the network queue.

5.7.2.4 xNetworkQueueIsEmpty()

```
BaseType_t xNetworkQueueIsEmpty (
             NetworkQueue_t * pxQueue )
```

Check if a network queue is empty.

This function checks if a network queue is empty by checking if the number of packets waiting in the queue is zero.

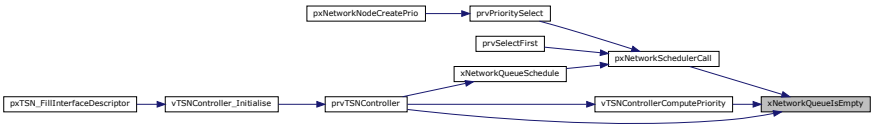
Parameters

pxQueue	A pointer to the network queue.

Returns

pdTRUE if the network queue is empty, pdFALSE otherwise.

Here is the caller graph for this function:

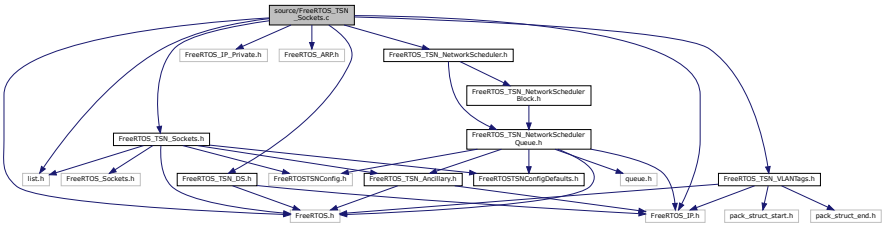


5.8 source/FreeRTOS_TSN_Sockets.c File Reference

FreeRTOS TSN Compatibility Layer - Socket Functions.

```
#include "FreeRTOS.h"
#include "list.h"
#include "FreeRTOS_IP.h"
#include "FreeRTOS_IP_Private.h"
#include "FreeRTOS_ARP.h"
#include "FreeRTOS_TSN_Sockets.h"
#include "FreeRTOS_TSN_NetworkScheduler.h"
#include "FreeRTOS_TSN_VLANTags.h"
#include "FreeRTOS_TSN_DS.h"
```

Include dependency graph for FreeRTOS_TSN_Sockets.c:



Macros

- #define tsnsocketSET_SOCKET_PORT(pxSocket, usPort) listSET_LIST_ITEM_VALUE((&((pxSocket)->xBoundSocketListItem)), (usPort))
- #define tsnsocketGET_SOCKET_PORT(pxSocket) listGET_LIST_ITEM_VALUE((&((pxSocket)->xBoundSocketListItem)))
- #define tsnsocketSOCKET_IS_BOUND(pxSocket) (listLIST_ITEM_CONTAINER(&(pxSocket)->xBoundSocketListItem) != NULL)

Functions

- void vInitialiseTSNSockets ()
- BaseType_t xSocketErrorQueueInsert (TSNSocket_t xTSNSocket, struct msghdr *pxMsgh)
- void vSocketFromPort (TickType_t xSearchKey, Socket_t *pxBaseSocket, TSNSocket_t *pxTSNSocket)

  Searches for a socket based on a given search key and retrieves the corresponding base socket and TSN socket.
- BaseType_t prvPrepareBufferUDPv4 (FreeRTOS_TSN_Socket_t *pxSocket, NetworkBufferDescriptor_t *pxBuf, BaseType_t xFlags, const struct freertos_sockaddr *pxDestinationAddress, BaseType_t xDestinationAddressLength)

  Prepare a buffer for sending UDPv4 packets.
- BaseType_t prvPrepareBufferUDPv6 (FreeRTOS_TSN_Socket_t *pxSocket, NetworkBufferDescriptor_t *pxBuf, BaseType_t xFlags, const struct freertos_sockaddr *pxDestinationAddress, BaseType_t xDestinationAddressLength)

  Prepares a UDPv6 buffer for transmission.
- void prvMoveToStartOfPayload (void **ppvBuf, size_t *puxSize)

  Moves the buffer pointer to the start of the payload and updates the payload size.
- TSNSocket_t FreeRTOS_TSN_socket (BaseType_t xDomain, BaseType_t xType, BaseType_t xProtocol)

  Creates a TSN socket.
- BaseType_t FreeRTOS_TSN_setsockopt (TSNSocket_t xSocket, int32_t lLevel, int32_t lOptionName, const void *pvOptionValue, size_t uxOptionLength)

  Set socket options for a TSN socket.
- BaseType_t FreeRTOS_TSN_bind (TSNSocket_t xSocket, struct freertos_sockaddr const *pxAddress, socklen_t xAddressLength)

  Binds a TSN socket to a specific address.
- BaseType_t FreeRTOS_TSN_closesocket (TSNSocket_t xSocket)

  Closes a TSN socket.
- int32_t FreeRTOS_TSN_sendto (TSNSocket_t xSocket, const void *pvBuffer, size_t uxTotalDataLength, BaseType_t xFlags, const struct freertos_sockaddr *pxDestinationAddress, socklen_t xDestinationAddressLength)

  Sends data to a TSN socket.
- int32_t FreeRTOS_TSN_recvmsg (TSNSocket_t xSocket, struct msghdr *pxMsghUser, BaseType_t xFlags)

  Receives a message from a TSN socket.
- int32_t FreeRTOS_TSN_recvfrom (TSNSocket_t xSocket, void *pvBuffer, size_t uxBufferLength, BaseType_t xFlags, struct freertos_sockaddr *pxSourceAddress, socklen_t *pxSourceAddressLength)

  Receive data from a TSN socket.

Variables

- static List_t xTSNBoundUDPSocketList

5.8.1 Detailed Description

FreeRTOS TSN Compatibility Layer - Socket Functions.

This file implements an alternative sockets API that works in parallel with +TCP sockets. This sockets have an extended set of features that are missing the original Addon but are essential when used within a time sensitive network.

5.8.2 Macro Definition Documentation

5.8.2.1 tsnsocketGET_SOCKET_PORT

```
#define tsnsocketGET_SOCKET_PORT(
              pxSocket ) listGET_LIST_ITEM_VALUE( ( &( ( pxSocket )->xBoundSocketListItem ) )
)
```

5.8.2.2 tsnsocketSET_SOCKET_PORT

```
#define tsnsocketSET_SOCKET_PORT(
              pxSocket,
              usPort ) listSET_LIST_ITEM_VALUE( ( &( ( pxSocket )->xBoundSocketListItem ) ), (
usPort ) )
```

5.8.2.3 tsnsocketSOCKET_IS_BOUND

```
#define tsnsocketSOCKET_IS_BOUND(
              pxSocket ) ( listLIST_ITEM_CONTAINER( &( pxSocket )->xBoundSocketListItem ) !=
NULL )
```

5.8.3 Function Documentation

5.8.3.1 FreeRTOS_TSN_bind()

```
BaseType_t FreeRTOS_TSN_bind (
             TSNSocket_t xSocket,
             struct freertos_sockaddr const * pxAddress,
             socklen_t xAddressLength )
```

Binds a TSN socket to a specific address.

This function binds a TSN socket to a specific address specified by `pxAddress`. The `xAddressLength` parameter specifies the length of the address structure.

Parameters

xSocket	The TSN socket to bind.
pxAddress	Pointer to the address structure.
xAddressLength	The length of the address structure.

Returns

If the socket is successfully bound, the function returns 0. Otherwise, it returns a negative value.

5.8.3.2 FreeRTOS_TSN_closesocket()

```
BaseType_t FreeRTOS_TSN_closesocket (
            TSNSocket_t xSocket )
```

Closes a TSN socket.

This function closes the specified TSN socket.

Parameters

xSocket	The TSN socket to be closed.

Returns

If the socket is successfully closed, the function returns 0. Otherwise, it returns an error code.

5.8.3.3 FreeRTOS_TSN_recvfrom()

```
int32_t FreeRTOS_TSN_recvfrom (
            TSNSocket_t xSocket,
            void * pvBuffer,
            size_t uxBufferLength,
            BaseType_t xFlags,
            struct freertos_sockaddr * pxSourceAddress,
            socklen_t * pxSourceAddressLength )
```

Receive data from a TSN socket.

This function receives data from a TSN socket and stores it in the provided buffer.

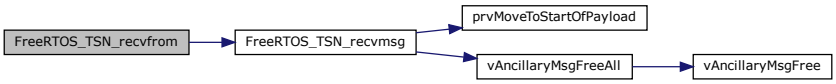
Parameters

xSocket	The TSN socket to receive data from.
pvBuffer	Pointer to the buffer where the received data will be stored.
uxBufferLength	The length of the buffer in bytes.
xFlags	Flags to control the behavior of the receive operation.
pxSourceAddress	Pointer to a structure that will hold the source address information.
pxSourceAddressLength	Pointer to the length of the source address structure.

Returns

The number of bytes received on success, or a negative error code on failure.

Here is the call graph for this function:



5.8.3.4 FreeRTOS_TSN_recvmsg()

```
int32_t FreeRTOS_TSN_recvmsg (
            TSNSocket_t xSocket,
            struct msghdr * pxMsghUser,
            BaseType_t xFlags )
```

Receives a message from a TSN socket.

This function receives a message from the specified TSN socket. It retrieves the message from the waiting packets list of the underlying base socket. If the `FREERTOS_MSG_ERRQUEUE` flag is set, it retrieves the message from the error queue of the TSN socket.

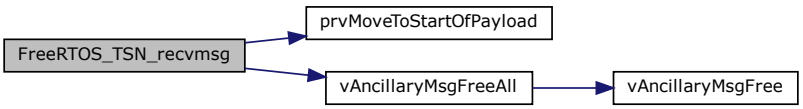
Parameters

xSocket	The TSN socket from which to receive the message.
pxMsghUser	Pointer to the `msghdr` structure that will hold the received message.
xFlags	Flags that control the behavior of the receive operation.

Returns

The length of the payload of the received message, or a negative error code if an error occurs.

Here is the call graph for this function:



Here is the caller graph for this function:

5.8.3.5 FreeRTOS_TSN_sendto()

```
int32_t FreeRTOS_TSN_sendto (
            TSNSocket_t xSocket,
            const void * pvBuffer,
            size_t uxTotalDataLength,
            BaseType_t xFlags,
            const struct freertos_sockaddr * pxDestinationAddress,
            socklen_t xDestinationAddressLength )
```

Sends data to a TSN socket.

This function sends data to a TSN socket specified by the `xSocket` parameter.

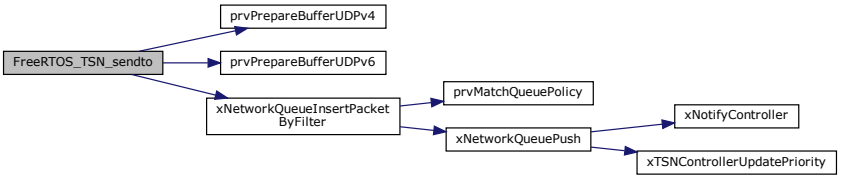
Parameters

xSocket	The TSN socket to send data to.
pvBuffer	Pointer to the data buffer containing the data to send.
uxTotalDataLength	The total length of the data to send.
xFlags	Flags to control the behavior of the send operation.
pxDestinationAddress	Pointer to the destination address structure.
xDestinationAddressLength	The length of the destination address structure.

Returns

The number of bytes sent on success, or a negative error code on failure.

Here is the call graph for this function:



5.8.3.6 FreeRTOS_TSN_setsockopt()

```
BaseType_t FreeRTOS_TSN_setsockopt (
             TSNSocket_t xSocket,
             int32_t lLevel,
             int32_t lOptionName,
             const void * pvOptionValue,
             size_t uxOptionLength )
```

Set socket options for a TSN socket.

This function sets various options for a TSN socket.

Parameters

xSocket	The TSN socket to set options for.
lLevel	The level at which the option is defined.
lOptionName	The name of the option to set.
pvOptionValue	A pointer to the value of the option.
uxOptionLength	The length of the option value.

Returns

pdPASS if the option is set successfully, or a negative value if an error occurs.

5.8.3.7 FreeRTOS_TSN_socket()

```
TSNSocket_t FreeRTOS_TSN_socket (
             BaseType_t xDomain,
             BaseType_t xType,
             BaseType_t xProtocol )
```

Creates a TSN socket.

This function creates a TSN socket with the specified domain, type, and protocol. Only UDP sockets are supported at the moment.

Parameters

xDomain	The domain of the socket.
xType	The type of the socket.
xProtocol	The protocol of the socket.

Returns

The created TSN socket, or FREERTOS_TSN_INVALID_SOCKET if an error occurred.

5.8.3.8 prvMoveToStartOfPayload()

```
void prvMoveToStartOfPayload (
            void ** ppvBuf,
            size_t * puxSize )
```

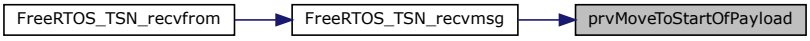
Moves the buffer pointer to the start of the payload and updates the payload size.

This function is used to move the buffer pointer to the start of the payload and update the payload size based on the frame type and protocol. It supports IPv4 and IPv6 frames with UDP, TCP, ICMP, ICMPv6, and IGMP protocols.

Parameters

in,out	*ppvBuf*	Pointer to the buffer pointer.
in,out	*puxSize*	Pointer to the payload size.

Here is the caller graph for this function:



5.8.3.9 prvPrepareBufferUDPv4()

```
BaseType_t prvPrepareBufferUDPv4 (
            FreeRTOS_TSN_Socket_t * pxSocket,
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xFlags,
            const struct freertos_sockaddr * pxDestinationAddress,
            BaseType_t xDestinationAddressLength )
```

Prepare a buffer for sending UDPv4 packets.

This function prepares a buffer for sending UDPv4 packets. It sets the necessary headers, including Ethernet, IP, and UDP headers, and performs ARP cache lookup to obtain the destination MAC address.

Parameters

pxSocket	The TSN socket.
pxBuf	The network buffer descriptor.
xFlags	The flags for the send operation.
pxDestinationAddress	The destination address.
xDestinationAddressLength	The length of the destination address.

Returns

pdPASS if the buffer is prepared successfully, pdFAIL otherwise.

Here is the caller graph for this function:

5.8.3.10 prvPrepareBufferUDPv6()

```
BaseType_t prvPrepareBufferUDPv6 (
            FreeRTOS_TSN_Socket_t * pxSocket,
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xFlags,
            const struct freertos_sockaddr * pxDestinationAddress,
            BaseType_t xDestinationAddressLength )
```

Prepares a UDPv6 buffer for transmission.

This function prepares a buffer for UDPv6 transmission by initializing the necessary data structures and copying the destination address into the packet header.

Parameters

pxSocket	The socket to which the buffer belongs.
pxBuf	The network buffer descriptor to be prepared.
xFlags	Flags to control the behavior of the function.
pxDestinationAddress	Pointer to the destination address structure.
xDestinationAddressLength	Length of the destination address.

Returns

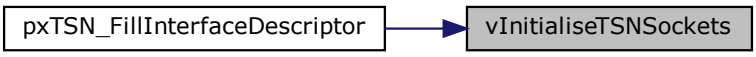
pdFAIL if the buffer preparation fails, pdPASS otherwise.

Here is the caller graph for this function:

5.8.3.11 vInitialiseTSNSockets()

```
void vInitialiseTSNSockets ( )
```

Here is the caller graph for this function:



5.8.3.12 vSocketFromPort()

```
void vSocketFromPort (
            TickType_t xSearchKey,
            Socket_t * pxBaseSocket,
            TSNSocket_t * pxTSNSocket )
```

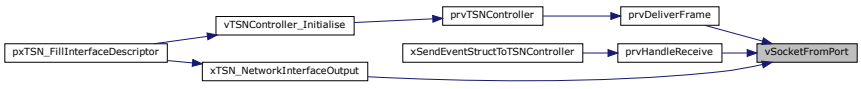
Searches for a socket based on a given search key and retrieves the corresponding base socket and TSN socket.

This function searches for a socket in the TSN bound UDP socket list based on the provided search key. If a matching socket is found, the corresponding TSN socket and base socket are retrieved.

Parameters

xSearchKey	The search key used to find the socket.
pxBaseSocket	Pointer to the base socket variable where the retrieved base socket will be stored.
pxTSNSocket	Pointer to the TSN socket variable where the retrieved TSN socket will be stored.

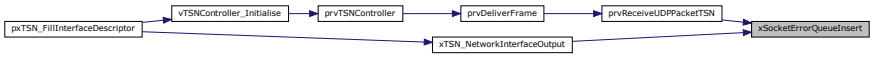
Here is the caller graph for this function:



5.8.3.13 xSocketErrorQueueInsert()

```
BaseType_t xSocketErrorQueueInsert (
            TSNSocket_t xTSNSocket,
            struct msghdr * pxMsgh )
```

Here is the caller graph for this function:



5.8.4 Variable Documentation

5.8.4.1 xTSNBoundUDPSocketList

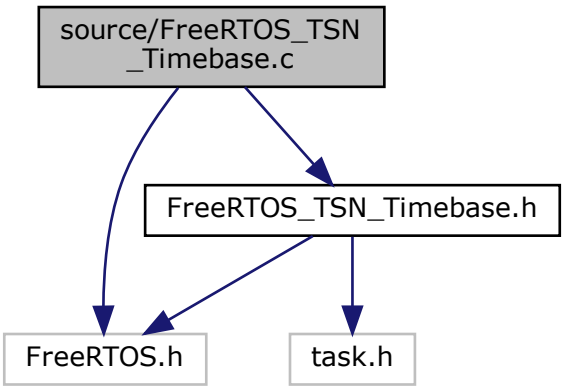
```
List_t xTSNBoundUDPSocketList  [static]
```

5.9 source/FreeRTOS_TSN_Timebase.c File Reference

Implementation of the FreeRTOS TSN Timebase module.

```
#include "FreeRTOS.h"
#include "FreeRTOS_TSN_Timebase.h"
```

Include dependency graph for FreeRTOS_TSN_Timebase.c:



Macros

- #define NS_IN_ONE_SEC (1000000000UL)

Functions

- BaseType_t xTimebaseHandleSet (TimebaseHandle_t *pxTimebase)

  Sets the timebase handle.
- void vTimebaseStart ()

  Starts the timebase.
- void vTimebaseSetTime (struct freertos_timespec *ts)

  Sets the time of the timebase.
- void vTimebaseGetTime (struct freertos_timespec *ts)

  Gets the current time of the timebase.
- void vTimebaseAdjTime (struct freertos_timespec *ts, BaseType_t xPositive)
- BaseType_t xTimebaseGetState ()

  Gets the state of the timebase.
- BaseType_t xTimespecSum (struct freertos_timespec *pxOut, struct freertos_timespec *pxOp1, struct freertos_timespec *pxOp2)

  Sums two timespec structures.
- BaseType_t xTimespecDiff (struct freertos_timespec *pxOut, struct freertos_timespec *pxOp1, struct freertos_timespec *pxOp2)

  Subtracts two timespec structures.
- BaseType_t xTimespecDiv (struct freertos_timespec *pxOut, struct freertos_timespec *pxOp1, BaseType_t xOp2)

  Divides a timespec structure by a scalar value.
- BaseType_t xTimespecCmp (struct freertos_timespec *pxOp1, struct freertos_timespec *pxOp2)

  Compares two timespec structures.

Variables

- static TimebaseHandle_t xTimebaseHandle
- static eTimebaseState_t xTimebaseState = eTimebaseNotInitialised

5.9.1 Detailed Description

Implementation of the FreeRTOS TSN Timebase module.

5.9.2 Macro Definition Documentation

5.9.2.1 NS_IN_ONE_SEC

```
#define NS_IN_ONE_SEC ( 1000000000UL )
```

5.9.3 Function Documentation

5.9.3.1 vTimebaseAdjTime()

```
void vTimebaseAdjTime (
            struct freertos_timespec * ts,
            BaseType_t xPositive )
```

5.9.3.2 vTimebaseGetTime()

```
void vTimebaseGetTime (
            struct freertos_timespec * ts )
```

Gets the current time of the timebase.

Parameters

ts	Pointer to the timespec structure to store the current time.

Here is the caller graph for this function:

5.9.3.3 vTimebaseSetTime()

```
void vTimebaseSetTime (
            struct freertos_timespec * ts )
```

Sets the time of the timebase.

Parameters

ts	Pointer to the timespec structure containing the time to be set.

5.9.3.4 vTimebaseStart()

```
void vTimebaseStart (
            void  )
```

Starts the timebase.

5.9.3.5 xTimebaseGetState()

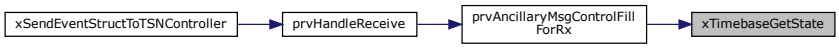
```
BaseType_t xTimebaseGetState (
            void  )
```

Gets the state of the timebase.

Returns

The state of the timebase.

Here is the caller graph for this function:



5.9.3.6 xTimebaseHandleSet()

```
BaseType_t xTimebaseHandleSet (
            TimebaseHandle_t * pxTimebase )
```

Sets the timebase handle.

Parameters

pxTimebase	Pointer to the timebase handle.

Returns

pdPASS if the timebase handle is set successfully, pdFAIL otherwise.

5.9.3.7 xTimespecCmp()

```
BaseType_t xTimespecCmp (
            struct freertos_timespec * pxOp1,
            struct freertos_timespec * pxOp2 )
```

Compares two timespec structures.

Parameters

pxOp1	Pointer to the first timespec structure.
pxOp2	Pointer to the second timespec structure.

Returns

1 if pxOp1 is greater than pxOp2, 0 if they are equal, -1 if pxOp1 is less than pxOp2.

5.9.3.8 xTimespecDiff()

```
BaseType_t xTimespecDiff (
            struct freertos_timespec * pxOut,
            struct freertos_timespec * pxOp1,
            struct freertos_timespec * pxOp2 )
```

Subtracts two timespec structures.

Parameters

pxOut	Pointer to the timespec structure to store the result.
pxOp1	Pointer to the first timespec structure.
pxOp2	Pointer to the second timespec structure.

Returns

pdPASS if the operation is successful, pdFAIL otherwise.

5.9.3.9 xTimespecDiv()

```
BaseType_t xTimespecDiv (
            struct freertos_timespec * pxOut,
            struct freertos_timespec * pxOp1,
            BaseType_t xOp2 )
```

Divides a timespec structure by a scalar value.

Parameters

pxOut	Pointer to the timespec structure to store the result.
pxOp1	Pointer to the timespec structure to be divided.
xOp2	The scalar value to divide by.

Returns

pdPASS if the operation is successful, pdFAIL otherwise.

Here is the call graph for this function:

5.9.3.10 xTimespecSum()

```
BaseType_t xTimespecSum (
            struct freertos_timespec * pxOut,
            struct freertos_timespec * pxOp1,
            struct freertos_timespec * pxOp2 )
```

Sums two timespec structures.

Parameters

pxOut	Pointer to the timespec structure to store the result.
pxOp1	Pointer to the first timespec structure.
pxOp2	Pointer to the second timespec structure.

Returns

pdPASS if the operation is successful, pdFAIL otherwise.

Here is the caller graph for this function:

5.9.4 Variable Documentation

5.9.4.1 xTimebaseHandle

```
TimebaseHandle_t xTimebaseHandle  [static]
```

5.9.4.2 xTimebaseState

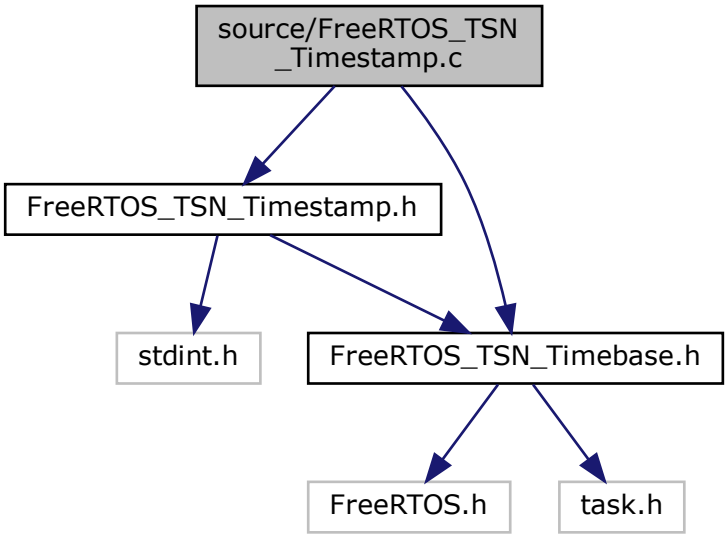
```
eTimebaseState_t xTimebaseState = eTimebaseNotInitialised  [static]
```

5.10 source/FreeRTOS_TSN_Timestamp.c File Reference

Implementation of the timestamping features.

```
#include "FreeRTOS_TSN_Timestamp.h"
#include "FreeRTOS_TSN_Timebase.h"
```

Include dependency graph for FreeRTOS_TSN_Timestamp.c:



Functions

- void vTimestampAcquireSoftware (struct freertos_timespec *ts)

  Acquires the software timestamp.

5.10.1 Detailed Description

Implementation of the timestamping features.

5.10.2 Function Documentation

5.10.2.1 vTimestampAcquireSoftware()

```
void vTimestampAcquireSoftware (
            struct freertos_timespec * ts )
```

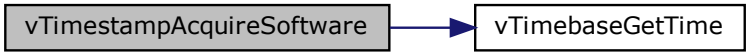
Acquires the software timestamp.

This function acquires the software timestamp by suspending all tasks, getting the time from the timebase, and then resuming all tasks.

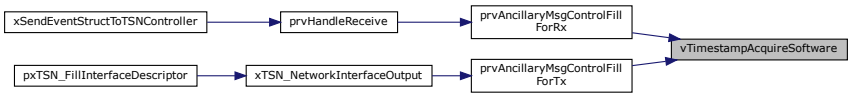
Parameters

ts	Pointer to the freertos_timespec structure where the acquired timestamp will be stored.

Here is the call graph for this function:



Here is the caller graph for this function:

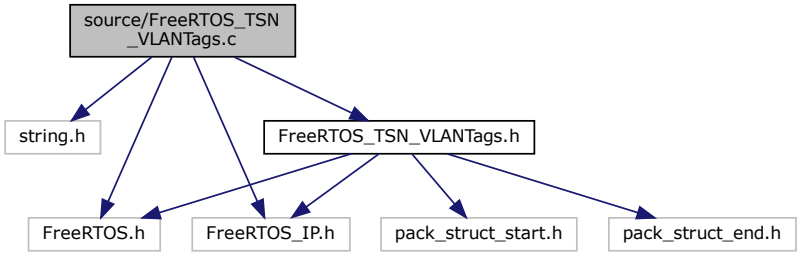


5.11 source/FreeRTOS_TSN_VLANTags.c File Reference

Implementation of functions for handling VLAN tags in FreeRTOS TSN Compatibility Layer.

```
#include <string.h>
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"
#include "FreeRTOS_TSN_VLANTags.h"
```

Include dependency graph for FreeRTOS_TSN_VLANTags.c:



Functions

- uint8_t ucGetNumberOfTags (NetworkBufferDescriptor_t ∗pxBuf)

  Get the number of VLAN tags in the given network buffer.
- struct xVLAN_TAG ∗ prvGetVLANSTag (NetworkBufferDescriptor_t ∗pxBuf, uint8_t ucNumTags)

  Get a pointer to the VLAN S-Tag in the network buffer.
- struct xVLAN_TAG ∗ prvGetVLANCTag (NetworkBufferDescriptor_t ∗pxBuf, uint8_t ucNumTags)

  Get a pointer to the VLAN C-Tag in the network buffer.
- struct xVLAN_TAG ∗ prvPrepareAndGetVLANCTag (NetworkBufferDescriptor_t ∗pxBuf)

  Prepare and get a pointer to the VLAN C-Tag in the network buffer.
- struct xVLAN_TAG ∗ prvPrepareAndGetVLANSTag (NetworkBufferDescriptor_t ∗pxBuf)

  Prepare and get a pointer to the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagGetPCP (NetworkBufferDescriptor_t ∗pxBuf)

  Get the Priority Code Point (PCP) from the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagGetDEI (NetworkBufferDescriptor_t ∗pxBuf)

  Get the Drop Eligible Indicator (DEI) from the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagGetVID (NetworkBufferDescriptor_t ∗pxBuf)

  Get the VLAN Identifier (VID) from the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagCheckClass (NetworkBufferDescriptor_t ∗pxBuf, BaseType_t xClass)

  Check if the VLAN S-Tag in the network buffer has a specific PCP value.
- BaseType_t xVLANCTagGetPCP (NetworkBufferDescriptor_t ∗pxBuf)

  Get the Priority Code Point (PCP) from the VLAN C-Tag in the network buffer.
- BaseType_t xVLANCTagGetDEI (NetworkBufferDescriptor_t ∗pxBuf)

  Get the Drop Eligible Indicator (DEI) from the VLAN C-Tag in the network buffer.
- BaseType_t xVLANCTagGetVID (NetworkBufferDescriptor_t ∗pxBuf)

  Get the VLAN Identifier (VID) from the VLAN C-Tag in the network buffer.
- BaseType_t xVLANCTagCheckClass (NetworkBufferDescriptor_t ∗pxBuf, BaseType_t xClass)

  Check if the VLAN C-Tag in the network buffer has a specific PCP value.
- BaseType_t xVLANCTagSetPCP (NetworkBufferDescriptor_t ∗pxBuf, BaseType_t xValue)

  Set the Priority Code Point (PCP) of the VLAN C-Tag in the network buffer.
- BaseType_t xVLANCTagSetDEI (NetworkBufferDescriptor_t ∗pxBuf, BaseType_t xValue)

  Set the Drop Eligible Indicator (DEI) of the VLAN C-Tag in the network buffer.
- BaseType_t xVLANCTagSetVID (NetworkBufferDescriptor_t ∗pxBuf, BaseType_t xValue)

  Set the VLAN Identifier (VID) of the VLAN C-Tag in the network buffer.
- BaseType_t xVLANSTagSetPCP (NetworkBufferDescriptor_t ∗pxBuf, BaseType_t xValue)

  Set the Priority Code Point (PCP) of the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagSetDEI (NetworkBufferDescriptor_t ∗pxBuf, BaseType_t xValue)

  Set the Drop Eligible Indicator (DEI) of the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagSetVID (NetworkBufferDescriptor_t ∗pxBuf, BaseType_t xValue)

  Set the VLAN Identifier (VID) of the VLAN S-Tag in the network buffer.

5.11.1 Detailed Description

Implementation of functions for handling VLAN tags in FreeRTOS TSN Compatibility Layer.

5.11.2 Function Documentation

5.11.2.1 prvGetVLANCTag()

```
struct xVLAN_TAG * prvGetVLANCTag (
            NetworkBufferDescriptor_t * pxBuf,
            uint8_t ucNumTags )
```

Get a pointer to the VLAN C-Tag in the network buffer.

This function calculates the offset of the VLAN C-Tag in the network buffer based on the number of VLAN tags and returns a pointer to it.

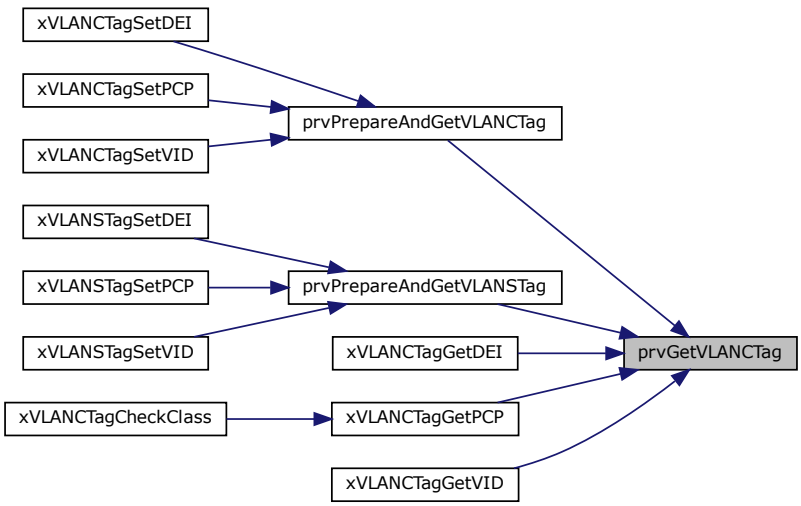
Parameters

pxBuf	Pointer to the network buffer descriptor.
ucNumTags	The number of VLAN tags in the network buffer.

Returns

Pointer to the VLAN C-Tag, or NULL if the VLAN C-Tag is not present.

Here is the caller graph for this function:



5.11.2.2 prvGetVLANSTag()

```
struct xVLAN_TAG * prvGetVLANSTag (
            NetworkBufferDescriptor_t * pxBuf,
            uint8_t ucNumTags )
```

Get a pointer to the VLAN S-Tag in the network buffer.

This function calculates the offset of the VLAN S-Tag in the network buffer based on the number of VLAN tags and returns a pointer to it.

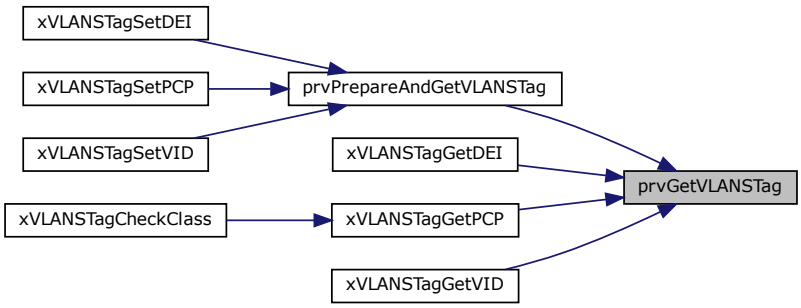
Parameters

pxBuf	Pointer to the network buffer descriptor.
ucNumTags	The number of VLAN tags in the network buffer.

Returns

Pointer to the VLAN S-Tag, or NULL if the VLAN S-Tag is not present.

Here is the caller graph for this function:



5.11.2.3 prvPrepareAndGetVLANCTag()

```
struct xVLAN_TAG * prvPrepareAndGetVLANCTag (
            NetworkBufferDescriptor_t * pxBuf )
```

Prepare and get a pointer to the VLAN C-Tag in the network buffer.

This function prepares the VLAN C-Tag in the network buffer if necessary and returns a pointer to it.

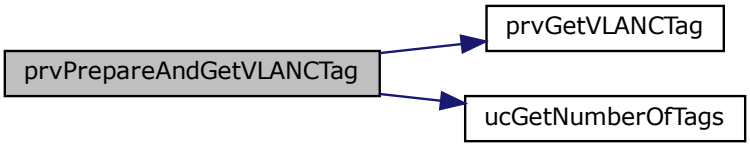
Parameters

pxBuf	Pointer to the network buffer descriptor.

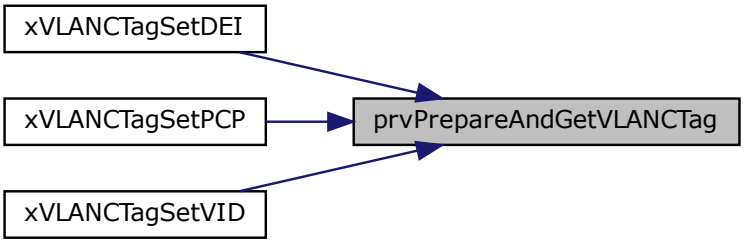
Returns

Pointer to the VLAN C-Tag, or NULL if the VLAN C-Tag cannot be prepared.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.4 prvPrepareAndGetVLANSTag()

```
struct xVLAN_TAG * prvPrepareAndGetVLANSTag (
            NetworkBufferDescriptor_t * pxBuf )
```

Prepare and get a pointer to the VLAN S-Tag in the network buffer.

This function prepares the VLAN S-Tag in the network buffer if necessary and returns a pointer to it.

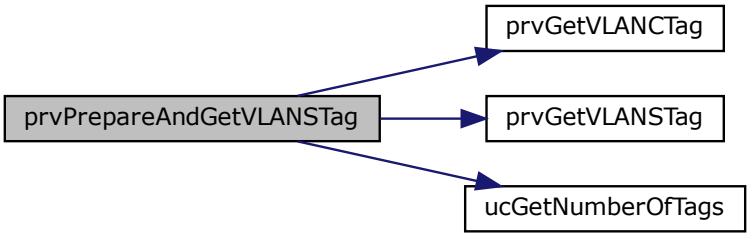
Parameters

pxBuf	Pointer to the network buffer descriptor.

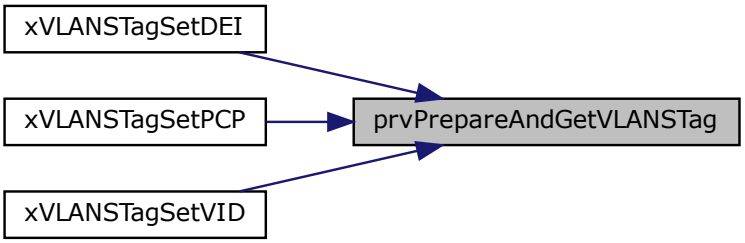
Returns

Pointer to the VLAN S-Tag, or NULL if the VLAN S-Tag cannot be prepared.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.5 ucGetNumberOfTags()

```
uint8_t ucGetNumberOfTags (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the number of VLAN tags in the given network buffer.

This function checks the Ethernet frame type in the network buffer and determines the number of VLAN tags present.

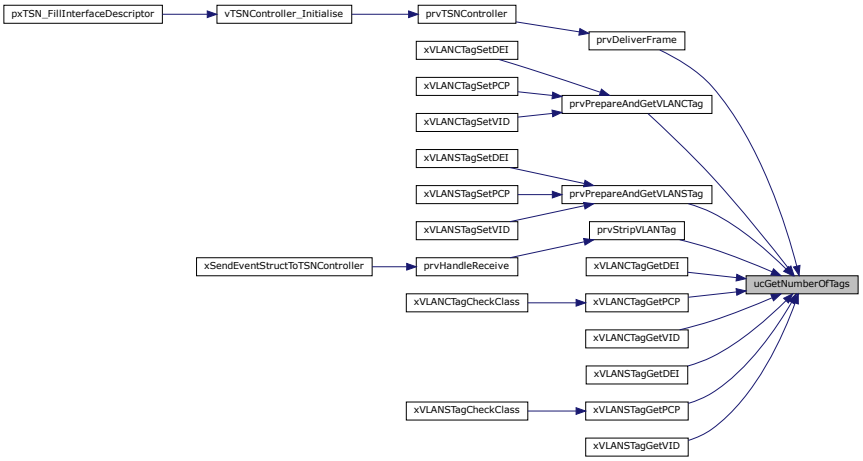
Parameters

pxBuf	Pointer to the network buffer descriptor.

Returns

The number of VLAN tags found in the network buffer.

Here is the caller graph for this function:



5.11.2.6 xVLANCTagCheckClass()

```
BaseType_t xVLANCTagCheckClass (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xClass )
```

Check if the VLAN C-Tag in the network buffer has a specific PCP value.

This function checks if the PCP value of the VLAN C-Tag in the network buffer matches the specified value.

Parameters

pxBuf	Pointer to the network buffer descriptor.
xClass	The PCP value to check against.

Returns

pdTRUE if the PCP value matches, pdFALSE otherwise.

Here is the call graph for this function:

5.11.2.7 xVLANCTagGetDEI()

```
BaseType_t xVLANCTagGetDEI (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the Drop Eligible Indicator (DEI) from the VLAN C-Tag in the network buffer.

This function retrieves the DEI value from the VLAN C-Tag in the network buffer.

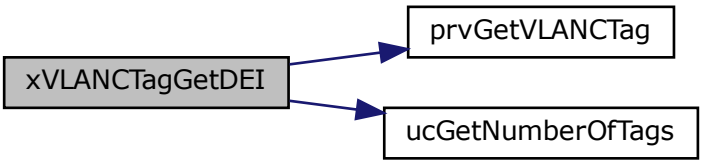
Parameters

pxBuf	Pointer to the network buffer descriptor.

Returns

The DEI value, or ~0 if the VLAN C-Tag is not present.

Here is the call graph for this function:



5.11.2.8 xVLANCTagGetPCP()

```
BaseType_t xVLANCTagGetPCP (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the Priority Code Point (PCP) from the VLAN C-Tag in the network buffer.

This function retrieves the PCP value from the VLAN C-Tag in the network buffer.

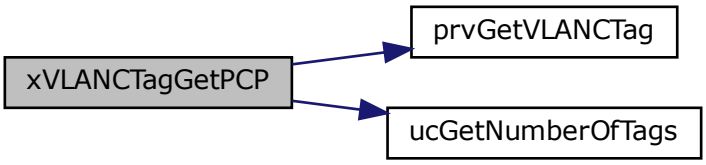
Parameters

pxBuf	Pointer to the network buffer descriptor.

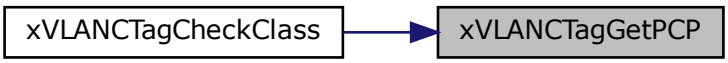
Returns

The PCP value, or ~0 if the VLAN C-Tag is not present.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.9 xVLANCTagGetVID()

```
BaseType_t xVLANCTagGetVID (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the VLAN Identifier (VID) from the VLAN C-Tag in the network buffer.

This function retrieves the VID value from the VLAN C-Tag in the network buffer.

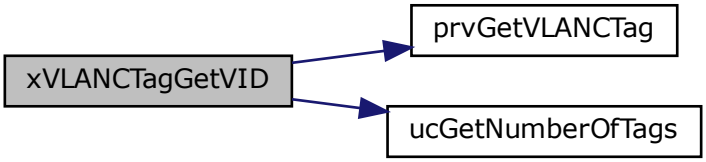
Parameters

pxBuf	Pointer to the network buffer descriptor.

Returns

The VID value, or ~0 if the VLAN C-Tag is not present.

Here is the call graph for this function:



5.11.2.10 xVLANCTagSetDEI()

```
BaseType_t xVLANCTagSetDEI (
             NetworkBufferDescriptor_t * pxBuf,
             BaseType_t xValue )
```

Set the Drop Eligible Indicator (DEI) of the VLAN C-Tag in the network buffer.

This function sets the DEI value of the VLAN C-Tag in the network buffer.

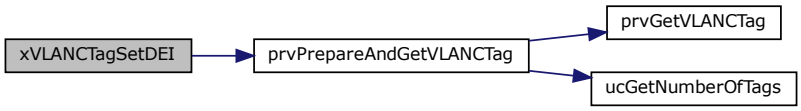
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The DEI value to set.

Returns

pdTRUE if the DEI value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.11.2.11 xVLANCTagSetPCP()

```
BaseType_t xVLANCTagSetPCP (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xValue )
```

Set the Priority Code Point (PCP) of the VLAN C-Tag in the network buffer.

This function sets the PCP value of the VLAN C-Tag in the network buffer.

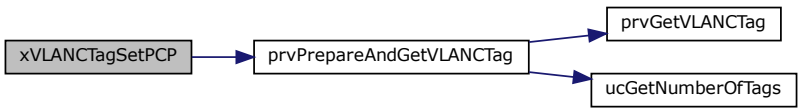
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The PCP value to set.

Returns

pdTRUE if the PCP value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.11.2.12 xVLANCTagSetVID()

```
BaseType_t xVLANCTagSetVID (
             NetworkBufferDescriptor_t * pxBuf,
             BaseType_t xValue )
```

Set the VLAN Identifier (VID) of the VLAN C-Tag in the network buffer.

This function sets the VID value of the VLAN C-Tag in the network buffer.

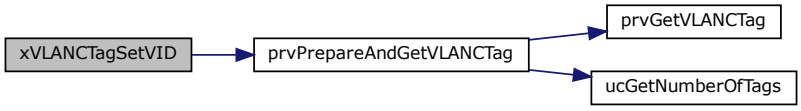
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The VID value to set.

Returns

pdTRUE if the VID value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.11.2.13 xVLANSTagCheckClass()

```
BaseType_t xVLANSTagCheckClass (
             NetworkBufferDescriptor_t * pxBuf,
             BaseType_t xClass )
```

Check if the VLAN S-Tag in the network buffer has a specific PCP value.

This function checks if the PCP value of the VLAN S-Tag in the network buffer matches the specified value.

Parameters

pxBuf	Pointer to the network buffer descriptor.
xClass	The PCP value to check against.

Returns

pdTRUE if the PCP value matches, pdFALSE otherwise.

Here is the call graph for this function:

5.11.2.14 xVLANSTagGetDEI()

```
BaseType_t xVLANSTagGetDEI (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the Drop Eligible Indicator (DEI) from the VLAN S-Tag in the network buffer.

This function retrieves the DEI value from the VLAN S-Tag in the network buffer.

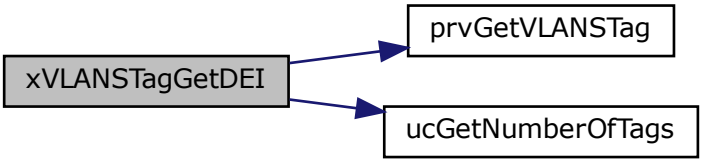
Parameters

pxBuf	Pointer to the network buffer descriptor.

Returns

The DEI value, or ~0 if the VLAN S-Tag is not present.

Here is the call graph for this function:



5.11.2.15 xVLANSTagGetPCP()

```
BaseType_t xVLANSTagGetPCP (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the Priority Code Point (PCP) from the VLAN S-Tag in the network buffer.

This function retrieves the PCP value from the VLAN S-Tag in the network buffer.

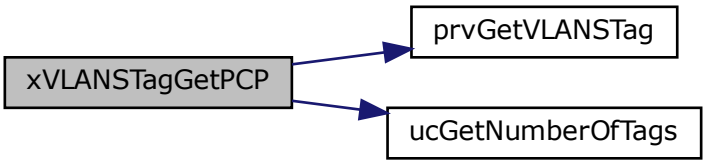
Parameters

pxBuf	Pointer to the network buffer descriptor.

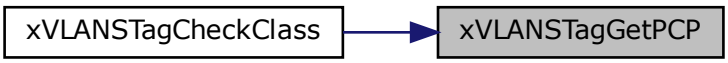
Returns

The PCP value, or ~0 if the VLAN S-Tag is not present.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.16 xVLANSTagGetVID()

```
BaseType_t xVLANSTagGetVID (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the VLAN Identifier (VID) from the VLAN S-Tag in the network buffer.

This function retrieves the VID value from the VLAN S-Tag in the network buffer.

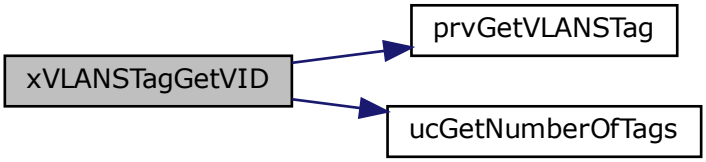
Parameters

pxBuf	Pointer to the network buffer descriptor.

Returns

The VID value, or ~0 if the VLAN S-Tag is not present.

Here is the call graph for this function:



5.11.2.17 xVLANSTagSetDEI()

```
BaseType_t xVLANSTagSetDEI (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xValue )
```

Set the Drop Eligible Indicator (DEI) of the VLAN S-Tag in the network buffer.

This function sets the DEI value of the VLAN S-Tag in the network buffer.

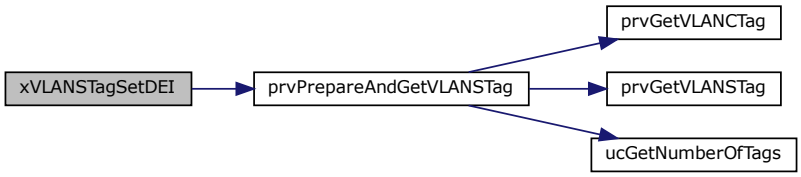
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The DEI value to set.

Returns

pdTRUE if the DEI value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.11.2.18 xVLANSTagSetPCP()

```
BaseType_t xVLANSTagSetPCP (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xValue )
```

Set the Priority Code Point (PCP) of the VLAN S-Tag in the network buffer.

This function sets the PCP value of the VLAN S-Tag in the network buffer.

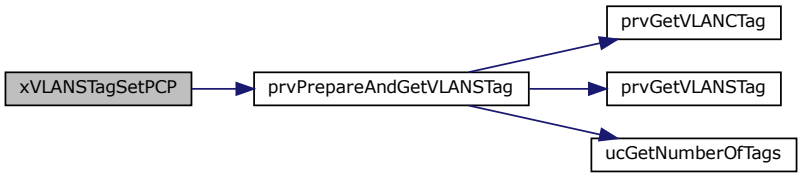
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The PCP value to set.

Returns

pdTRUE if the PCP value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.11.2.19 xVLANSTagSetVID()

```
BaseType_t xVLANSTagSetVID (
             NetworkBufferDescriptor_t * pxBuf,
             BaseType_t xValue )
```

Set the VLAN Identifier (VID) of the VLAN S-Tag in the network buffer.

This function sets the VID value of the VLAN S-Tag in the network buffer.

Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The VID value to set.

Returns

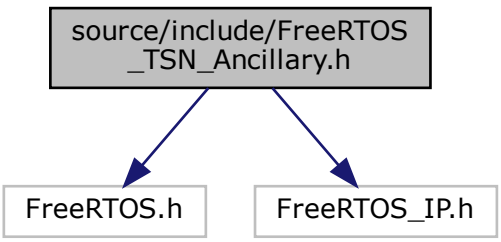
pdTRUE if the VID value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:

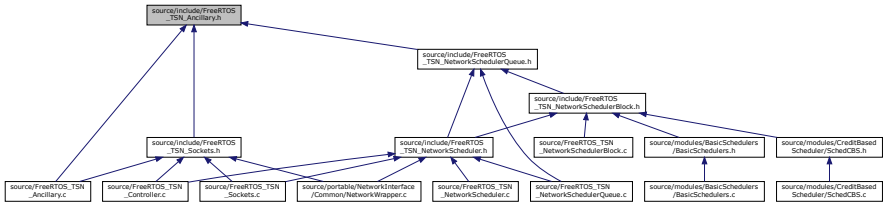
5.12 source/include/FreeRTOS_TSN_Ancillary.h File Reference

```
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"
```

Include dependency graph for FreeRTOS_TSN_Ancillary.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct iovec
- struct msghdr
- struct cmsghdr
- struct sock_extended_err

Macros

- #define pdFREERTOS_ERRNO_ENOMSG 42
- #define SO_EE_ORIGIN_NONE 0
- #define SO_EE_ORIGIN_LOCAL 1
- #define SO_EE_ORIGIN_ICMP 2
- #define SO_EE_ORIGIN_ICMP6 3
- #define SO_EE_ORIGIN_TXSTATUS 4
- #define SO_EE_ORIGIN_ZEROCOPY 5
- #define SO_EE_ORIGIN_TXTIME 6
- #define SO_EE_ORIGIN_TIMESTAMPING SO_EE_ORIGIN_TXSTATUS
- #define CMSG_ALIGN(len) (((len) + sizeof(long) - 1) & ~(sizeof(long) - 1))
- #define CMSG_DATA(cmsg) ((void *) ((char *) (cmsg) + CMSG_ALIGN(sizeof(struct cmsghdr))))
- #define CMSG_SPACE(len) (CMSG_ALIGN(sizeof(struct cmsghdr)) + CMSG_ALIGN(len))
- #define CMSG_LEN(len) (CMSG_ALIGN(sizeof(struct cmsghdr)) + (len))
- #define __CMSG_FIRSTHDR(ctl, len)
- #define CMSG_FIRSTHDR(msg) __CMSG_FIRSTHDR((msg)->msg_control, (msg)->msg_controllen)
- #define CMSG_NXTHDR(mhdr, cmsg) __CMSG_NXTHDR((mhdr)->msg_control, (mhdr)->msg_↩ controllen, (cmsg))

Functions

- struct cmsghdr * __CMSG_NXTHDR (void *ctl, size_t size, struct cmsghdr *cmsg)

  Aligns the size of a control message buffer.
- struct msghdr * pxAncillaryMsgMalloc ()

  Allocates memory for a new msghdr structure.
- void vAncillaryMsgFree (struct msghdr *pxMsgh)

  Frees the memory allocated for an ancillary message.
- void vAncillaryMsgFreeAll (struct msghdr *pxMsgh)

  Frees a msghdr.
- BaseType_t xAncillaryMsgFillName (struct msghdr *pxMsgh, IP_Address_t *xAddr, uint16_t usPort, Base↩ Type_t xFamily)

  Fills in the name field of a message header structure with the given IP address, port, and family.

- void vAncillaryMsgFreeName (struct msghdr *pxMsgh)

  Frees the memory allocated for the name field in the given msghdr structure.
- BaseType_t xAncillaryMsgFillPayload (struct msghdr *pxMsgh, uint8_t *pucBuffer, size_t uxLength)

  Fills the payload of an ancillary message.
- void vAncillaryMsgFreePayload (struct msghdr *pxMsgh)

  Frees the payload of an ancillary message.
- BaseType_t xAncillaryMsgControlFill (struct msghdr *pxMsgh, struct cmsghdr *pxCmsgVec, void **pv↩ DataVec, size_t *puxDataLenVec, size_t uxNumBuffers)

  Fills the ancillary message control structure with data.
- BaseType_t xAncillaryMsgControlFillSingle (struct msghdr *pxMsgh, struct cmsghdr *pxCmsg, void *pvData, size_t puxDataLen)
- void vAncillaryMsgFreeControl (struct msghdr *pxMsgh)

  Frees the memory allocated for the ancillary message control data.

5.12.1 Macro Definition Documentation

5.12.1.1 __CMSG_FIRSTHDR

```
#define __CMSG_FIRSTHDR(
              ctl,
              len )
```

Value:

```
    ( ( len ) >= sizeof( struct cmsghdr ) ?  \
      ( struct cmsghdr * ) ( ctl ) :                \
      ( struct cmsghdr * ) NULL )
```

5.12.1.2 CMSG_ALIGN

```
#define CMSG_ALIGN(
              len ) ( ( ( len ) + sizeof( long ) - 1 ) & ~( sizeof( long ) - 1 ) )
```

5.12.1.3 CMSG_DATA

```
#define CMSG_DATA(
              cmsg ) ( ( void * ) ( ( char * ) ( cmsg ) + CMSG_ALIGN( sizeof( struct cmsghdr )
) ) )
```

5.12.1.4 CMSG_FIRSTHDR

```
#define CMSG_FIRSTHDR(
              msg ) __CMSG_FIRSTHDR( ( msg )->msg_control, ( msg )->msg_controllen )
```

5.12.1.5 CMSG_LEN

```
#define CMSG_LEN(
              len ) ( CMSG_ALIGN( sizeof( struct cmsghdr ) ) + ( len ) )
```

5.12.1.6 CMSG_NXTHDR

```
#define CMSG_NXTHDR(
              mhdr,
              cmsg ) __CMSG_NXTHDR( ( mhdr )->msg_control, ( mhdr )->msg_controllen, ( cmsg )
)
```

5.12.1.7 CMSG_SPACE

```
#define CMSG_SPACE(
              len ) ( CMSG_ALIGN( sizeof( struct cmsghdr ) ) + CMSG_ALIGN( len ) )
```

5.12.1.8 pdFREERTOS_ERRNO_ENOMSG

```
#define pdFREERTOS_ERRNO_ENOMSG 42
```

5.12.1.9 SO_EE_ORIGIN_ICMP

```
#define SO_EE_ORIGIN_ICMP 2
```

5.12.1.10 SO_EE_ORIGIN_ICMP6

```
#define SO_EE_ORIGIN_ICMP6 3
```

5.12.1.11 SO_EE_ORIGIN_LOCAL

```
#define SO_EE_ORIGIN_LOCAL 1
```

5.12.1.12 SO_EE_ORIGIN_NONE

```
#define SO_EE_ORIGIN_NONE 0
```

5.12.1.13 SO_EE_ORIGIN_TIMESTAMPING

```
#define SO_EE_ORIGIN_TIMESTAMPING SO_EE_ORIGIN_TXSTATUS
```

5.12.1.14 SO_EE_ORIGIN_TXSTATUS

```
#define SO_EE_ORIGIN_TXSTATUS 4
```

5.12.1.15 SO_EE_ORIGIN_TXTIME

```
#define SO_EE_ORIGIN_TXTIME 6
```

5.12.1.16 SO_EE_ORIGIN_ZEROCOPY

```
#define SO_EE_ORIGIN_ZEROCOPY 5
```

5.12.2 Function Documentation

5.12.2.1 __CMSG_NXTHDR()

```
struct cmsghdr * __CMSG_NXTHDR (
            void * ctl,
            size_t size,
            struct cmsghdr * cmsg )
```

Aligns the size of a control message buffer.

5.12.2.2 pxAncillaryMsgMalloc()

```
struct msghdr * pxAncillaryMsgMalloc ( )
```

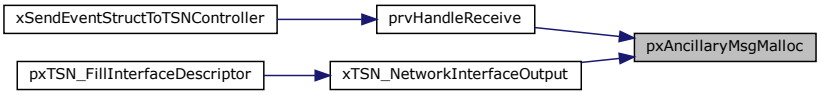
Allocates memory for a new msghdr structure.

This function allocates memory for a new msghdr structure using the pvPortMalloc function. The allocated memory is then initialized with zeros using the memset function.

Returns

A pointer to the newly allocated msghdr structure.

Here is the caller graph for this function:



5.12.2.3 vAncillaryMsgFree()

```
void vAncillaryMsgFree (
            struct msghdr * pxMsgh )
```

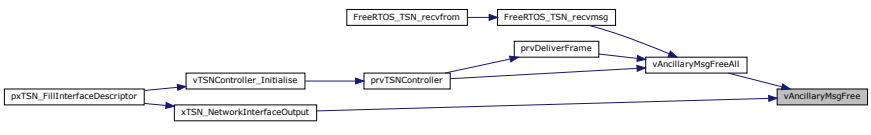
Frees the memory allocated for an ancillary message.

This function frees the memory allocated for the given ancillary message.

Parameters

pxMsgh	Pointer to the `msghdr` structure representing the ancillary message.

Here is the caller graph for this function:



5.12.2.4 vAncillaryMsgFreeAll()

```
void vAncillaryMsgFreeAll (
            struct msghdr * pxMsgh )
```

Frees a msghdr.

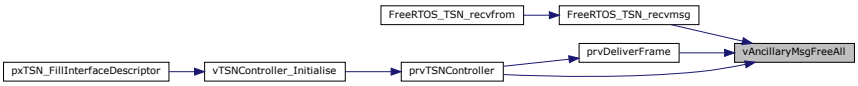
This will free all the non null members of the msghdr. In order to make sense it should always be used on a msghdr created using pxAncillaryMsgMalloc(), which takes the duty of initializing the struct to zero. Also note that this frees the iovec array, but not the iov_base buffers.

Parameters

pxMsgh	Pointer to msghdr to free

Here is the call graph for this function:

Here is the caller graph for this function:



5.12.2.5 vAncillaryMsgFreeControl()

```
void vAncillaryMsgFreeControl (
            struct msghdr * pxMsgh )
```

Frees the memory allocated for the ancillary message control data.

This function frees the memory allocated for the ancillary message control data pointed to by the `msg_control` member of the `msghdr` structure.

Parameters

pxMsgh	Pointer to the `msghdr` structure.

5.12.2.6 vAncillaryMsgFreeName()

```
void vAncillaryMsgFreeName (
            struct msghdr * pxMsgh )
```

Frees the memory allocated for the name field in the given msghdr structure.

This function frees the memory allocated for the name field in the provided msghdr structure.

Parameters

pxMsgh	Pointer to the msghdr structure.

5.12.2.7 vAncillaryMsgFreePayload()

```
void vAncillaryMsgFreePayload (
            struct msghdr * pxMsgh )
```

Frees the payload of an ancillary message.

This function frees the memory allocated for the payload of an ancillary message. The caller must ensure that the array is not empty before calling this function.

Parameters

pxMsgh	Pointer to the msghdr structure representing the ancillary message.

5.12.2.8 xAncillaryMsgControlFill()

```
BaseType_t xAncillaryMsgControlFill (
            struct msghdr * pxMsgh,
            struct cmsghdr * pxCmsgVec,
            void ** ppvDataVec,
            size_t * puxDataLenVec,
            size_t uxNumBuffers )
```

Fills the ancillary message control structure with data.

This function fills the ancillary message control structure with data provided in the input parameters.

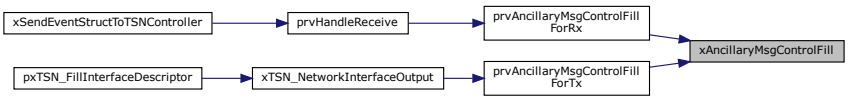
Parameters

pxMsgh	A pointer to the msghdr structure representing the message header.
pxCmsgVec	A pointer to the cmsghdr structure representing the control message vector.
ppvDataVec	An array of void pointers representing the data vector.
puxDataLenVec	An array of size_t values representing the data length vector.
uxNumBuffers	The number of buffers in the data vector.

Returns

pdTRUE if the ancillary message control structure is successfully filled, pdFAIL otherwise.

Here is the caller graph for this function:



5.12.2.9 xAncillaryMsgControlFillSingle()

```
BaseType_t xAncillaryMsgControlFillSingle (
             struct msghdr * pxMsgh,
             struct cmsghdr * pxCmsg,
             void * pvData,
             size_t puxDataLen )
```

5.12.2.10 xAncillaryMsgFillName()

```
BaseType_t xAncillaryMsgFillName (
             struct msghdr * pxMsgh,
             IP_Address_t * xAddr,
             uint16_t usPort,
             BaseType_t xFamily )
```

Fills in the name field of a message header structure with the given IP address, port, and family.

This function is used to fill in the name field of a message header structure with the given IP address, port, and family. The name field is used to specify the source or destination address of a socket.

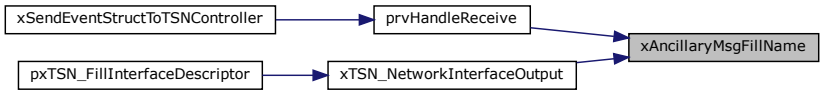
Parameters

pxMsgh	A pointer to the message header structure.
xAddr	A pointer to the IP address to be filled in the name field.
usPort	The port number to be filled in the name field.
xFamily	The address family to be filled in the name field.

Returns

pdPASS if the name field is successfully filled, pdFAIL otherwise.

Here is the caller graph for this function:



5.12.2.11 xAncillaryMsgFillPayload()

```
BaseType_t xAncillaryMsgFillPayload (
             struct msghdr * pxMsgh,
             uint8_t * pucBuffer,
             size_t uxLength )
```

Fills the payload of an ancillary message.

This function fills the payload of an ancillary message with the provided buffer and length.

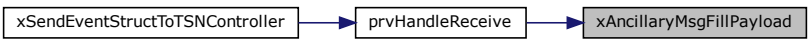
Parameters

pxMsgh	Pointer to the msghdr structure representing the ancillary message.
pucBuffer	Pointer to the buffer containing the payload data.
uxLength	Length of the payload data in bytes.

Returns

pdPASS if the payload was successfully filled, pdFAIL otherwise.

Here is the caller graph for this function:



5.13 FreeRTOS_TSN_Ancillary.h

Go to the documentation of this file.

```
1 #ifndef FREERTOS_TSN_ANCILLARY_H
```

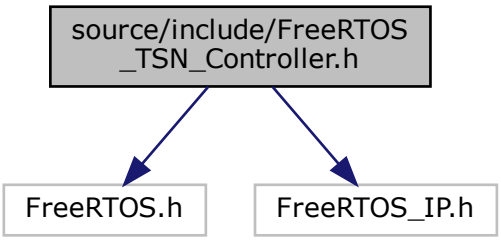
```
2 #define FREERTOS_TSN_ANCILLARY_H
3
4 #include "FreeRTOS.h"
5
6 #include "FreeRTOS_IP.h"
7
8 struct iovec         /* Scatter/gather array items */
9 {
10     void * iov_base; /* Starting address */
11     size_t iov_len;  /* Number of bytes to transfer */
12 };
13
14 struct msghdr
15 {
16     void * msg_name;        /* optional address */
17     socklen_t msg_namelen;  /* size of address */
18     struct iovec * msg_iov; /* scatter/gather array */
19     size_t msg_iovlen;      /* # elements in msg_iov */
20     void * msg_control;     /* ancillary data, see below */
21     size_t msg_controllen;  /* ancillary data buffer len */
22     int msg_flags;          /* flags on received message */
23 };
24
25 struct cmsghdr
26 {
27     socklen_t cmsg_len; /* data byte count, including header */
28     int cmsg_level;     /* originating protocol */
29     int cmsg_type;      /* protocol-specific type */
30     /* followed by unsigned char cmsg_data[]; */
31 };
32
33 struct sock_extended_err
34 {
35     uint32_t ee_errno; /* Error number */
36     uint8_t ee_origin; /* Where the error originated */
37     uint8_t ee_type;   /* Type */
38     uint8_t ee_code;   /* Code */
39     uint8_t ee_pad;    /* Padding */
40     uint32_t ee_info;  /* Additional information */
41     uint32_t ee_data;  /* Other data */
42     /* More data may follow */
43 };
44
45 #ifndef pdFREERTOS_ERRNO_ENOMSG
46 /* This is somehow missing from projdefs.h */
47 #define pdFREERTOS_ERRNO_ENOMSG    42
48 #endif
49
50 #define SO_EE_ORIGIN_NONE              0
51 #define SO_EE_ORIGIN_LOCAL             1
52 #define SO_EE_ORIGIN_ICMP              2
53 #define SO_EE_ORIGIN_ICMP6             3
54 #define SO_EE_ORIGIN_TXSTATUS          4
55 #define SO_EE_ORIGIN_ZEROCOPY          5
56 #define SO_EE_ORIGIN_TXTIME            6
57 #define SO_EE_ORIGIN_TIMESTAMPING      SO_EE_ORIGIN_TXSTATUS
58
59 #define CMSG_ALIGN( len )     ( ( ( len ) + sizeof( long ) - 1 ) & ~( sizeof( long ) - 1 ) )
60
61 #define CMSG_DATA( cmsg )     ( ( void * ) ( ( char * ) ( cmsg ) + CMSG_ALIGN( sizeof( struct cmsghdr ) )
    ) )
62
63 #define CMSG_SPACE( len )     ( CMSG_ALIGN( sizeof( struct cmsghdr ) ) + CMSG_ALIGN( len ) )
64
65 #define CMSG_LEN( len )       ( CMSG_ALIGN( sizeof( struct cmsghdr ) ) + ( len ) )
66
67 #define __CMSG_FIRSTHDR( ctl, len )         \
68 ( ( len ) >= sizeof( struct cmsghdr ) ?  \
69 ( struct cmsghdr * ) ( ctl ) :               \
70 ( struct cmsghdr * ) NULL )
71
72 #define CMSG_FIRSTHDR( msg )     __CMSG_FIRSTHDR( ( msg )->msg_control, ( msg )->msg_controllen )
73
74 struct cmsghdr * __CMSG_NXTHDR( void * ctl,
75                                 size_t size,
76                                 struct cmsghdr * cmsg );
77
78 #define CMSG_NXTHDR( mhdr, cmsg )     __CMSG_NXTHDR( ( mhdr )->msg_control, ( mhdr )->msg_controllen, (
    cmsg ) )
79
80
81 struct msghdr * pxAncillaryMsgMalloc();
82
83 void vAncillaryMsgFree( struct msghdr * pxMsgh );
84
85 void vAncillaryMsgFreeAll( struct msghdr * pxMsgh );
86
```

```
87 BaseType_t xAncillaryMsgFillName( struct msghdr * pxMsgh,
88                                    IP_Address_t * xAddr,
89                                    uint16_t usPort,
90                                    BaseType_t xFamily );
91
92 void vAncillaryMsgFreeName( struct msghdr * pxMsgh );
93
94 BaseType_t xAncillaryMsgFillPayload( struct msghdr * pxMsgh,
95                                      uint8_t * pucBuffer,
96                                      size_t uxLength );
97
98 void vAncillaryMsgFreePayload( struct msghdr * pxMsgh );
99
100 BaseType_t xAncillaryMsgControlFill( struct msghdr * pxMsgh,
101                                      struct cmsghdr * pxCmsgVec,
102                                      void ** pvDataVec,
103                                      size_t * puxDataLenVec,
104                                      size_t uxNumBuffers );
105
106 BaseType_t xAncillaryMsgControlFillSingle( struct msghdr * pxMsgh,
107                                            struct cmsghdr * pxCmsg,
108                                            void * pvData,
109                                            size_t puxDataLen );
110
111 void vAncillaryMsgFreeControl( struct msghdr * pxMsgh );
112
113 #endif /* FREERTOS_TSN_ANCILLARY_H */
```

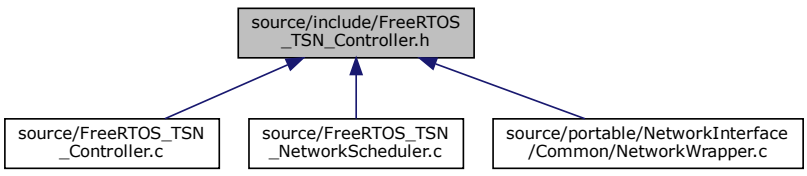
5.14 source/include/FreeRTOS_TSN_Controller.h File Reference

```
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"
```

Include dependency graph for FreeRTOS_TSN_Controller.h:



This graph shows which files directly or indirectly include this file:



Functions

- BaseType_t xNotifyController ()

  Function to notify the TSN Controller task.
- void vTSNControllerComputePriority (void)

  Function to compute the priority of the TSN Controller task.
- BaseType_t xTSNControllerUpdatePriority (UBaseType_t uxPriority)

  Function to update the priority of the TSN Controller task.
- void vTSNController_Initialise (void)

  Function to initialize the TSN Controller task.
- BaseType_t xIsCallingFromTSNController (void)

  Function to check if the caller task is the TSN Controller task.

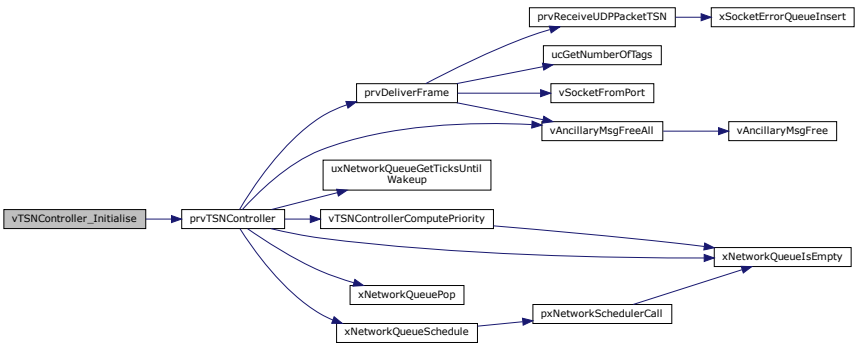
5.14.1 Function Documentation

5.14.1.1 vTSNController_Initialise()

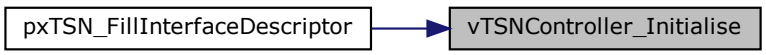
```
void vTSNController_Initialise (
            void  )
```

Function to initialize the TSN Controller task.

This function creates the TSN Controller task and sets its priority. Here is the call graph for this function:



Here is the caller graph for this function:

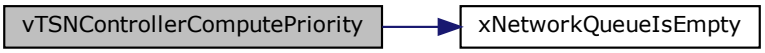


5.14.1.2 vTSNControllerComputePriority()

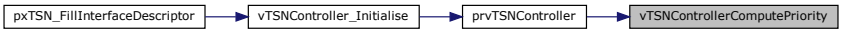
```
void vTSNControllerComputePriority (
            void  )
```

Function to compute the priority of the TSN Controller task.

The priority of the TSN controller is the maximum IPV among all the queues which has pending messages. Here is the call graph for this function:



Here is the caller graph for this function:



5.14.1.3 xIsCallingFromTSNController()

```
BaseType_t xIsCallingFromTSNController (
            void  )
```

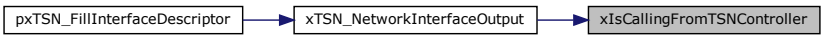
Function to check if the caller task is the TSN Controller task.

This function checks if the caller task is the TSN Controller task.

Returns

pdTRUE if the current task is the TSN Controller task, pdFALSE otherwise

Here is the caller graph for this function:



5.14.1.4 xNotifyController()

```
BaseType_t xNotifyController ( )
```

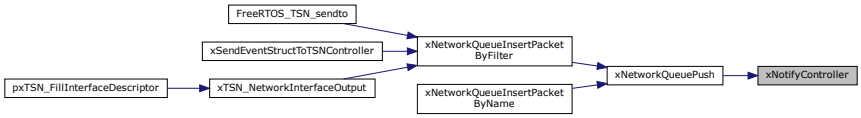
Function to notify the TSN Controller task.

This function notifies the TSN Controller task to wake up and process pending network packets or events.

Returns

pdTRUE if the notification is sent successfully, pdFALSE otherwise

Here is the caller graph for this function:



5.14.1.5 xTSNControllerUpdatePriority()

```
BaseType_t xTSNControllerUpdatePriority (
            UBaseType_t uxPriority )
```

Function to update the priority of the TSN Controller task.

This function updates the priority of the TSN Controller task if the new priority is higher than the current priority.

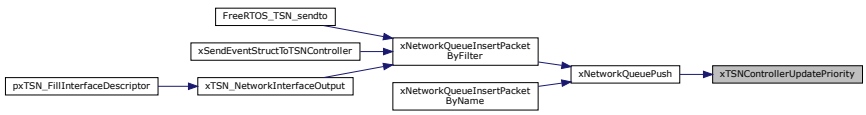
Parameters

in	*uxPriority*	New priority for the TSN Controller task

Returns

pdTRUE if the priority is updated, pdFALSE otherwise

Here is the caller graph for this function:



5.15 FreeRTOS_TSN_Controller.h

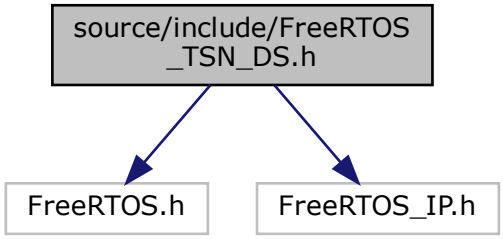
Go to the documentation of this file.

```
1 #ifndef FREERTOS_TSN_CONTROLLER
2 #define FREERTOS_TSN_CONTROLLER
3
4 #include "FreeRTOS.h"
5 #include "FreeRTOS_IP.h"
6
7 BaseType_t xNotifyController();
8
9 void vTSNControllerComputePriority( void );
10
11 BaseType_t xTSNControllerUpdatePriority( UBaseType_t uxPriority );
12
13 void vTSNController_Initialise( void );
14
15 BaseType_t xIsCallingFromTSNController( void );
16
17 #endif /* ifndef FREERTOS_TSN_CONTROLLER */
```

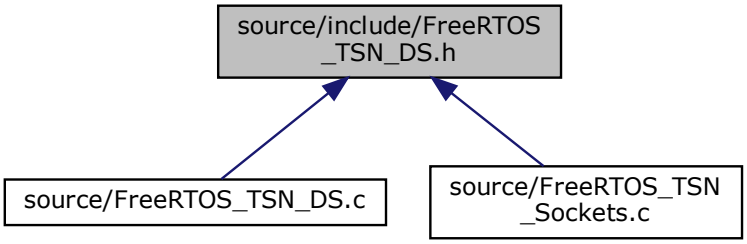
5.16 source/include/FreeRTOS_TSN_DS.h File Reference

```
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"
```

Include dependency graph for FreeRTOS_TSN_DS.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define diffservCLASS_DF (0)
- #define diffservCLASS_CSx(x) ((0 <= x && x <= 7) ? (8 * x) : diffservCLASS_DF)
- #define diffservCLASS_AFxy(x, y) ((1 <= x && x <= 4 && 1 <= y && y <= 3) ? (8 * x + 2 * y) : diffservCLASS_DF)
- #define diffservCLASS_LE (1)
- #define diffservCLASS_EF (46)
- #define diffservCLASS_DSCP_CUSTOM(x) (x & 0x3F)
- #define diffservGET_DSCLASS_IPv4(pxIPHeader) (pxIPHeader->ucDifferentiatedServicesCode >> 2)
- #define diffservGET_DSCLASS_IPv6(pxIPHeader) (((pxIPHeader->ucVersionTrafficClass & 0xF) << 2) | ((pxIPHeader->ucTrafficClassFlow & 0xC0) >> 6))
- #define diffservSET_DSCLASS_IPv4(pxIPHeader, ucValue)
- #define diffservSET_DSCLASS_IPv6(pxIPHeader, ucValue)

Functions

- uint8_t ucDSClassGet (NetworkBufferDescriptor_t *pxBuf)

  Retrieves the DiffServ class from the given network buffer.
- BaseType_t xDSClassSet (NetworkBufferDescriptor_t *pxBuf, uint8_t ucValue)

  Sets the DiffServ class for the given network buffer.

5.16.1 Macro Definition Documentation

5.16.1.1 diffservCLASS_AFxy

```
#define diffservCLASS_AFxy(
              x,
              y ) ( ( 1 <= x && x <= 4 && 1 <= y && y <= 3 ) ?  ( 8 * x + 2 * y ) :  diffservCLASS_DF
)
```

5.16.1.2 diffservCLASS_CSx

```
#define diffservCLASS_CSx(
              x ) ( ( 0 <= x && x <= 7 ) ?  ( 8 * x ) :  diffservCLASS_DF )
```

5.16.1.3 diffservCLASS_DF

```
#define diffservCLASS_DF ( 0 )
```

5.16.1.4 diffservCLASS_DSCP_CUSTOM

```
#define diffservCLASS_DSCP_CUSTOM(
             x ) ( x & 0x3F )
```

max 6 bits (0b11 1111)

5.16.1.5 diffservCLASS_EF

```
#define diffservCLASS_EF ( 46 )
```

5.16.1.6 diffservCLASS_LE

```
#define diffservCLASS_LE ( 1 )
```

5.16.1.7 diffservGET_DSCLASS_IPv4

```
#define diffservGET_DSCLASS_IPv4(
             pxIPHeader )  ( pxIPHeader->ucDifferentiatedServicesCode >> 2 )
```

5.16.1.8 diffservGET_DSCLASS_IPv6

```
#define diffservGET_DSCLASS_IPv6(
             pxIPHeader )  ( ( ( pxIPHeader->ucVersionTrafficClass & 0xF ) << 2 ) | ( ( px↩
IPHeader->ucTrafficClassFlow & 0xC0 ) >> 6 ) )
```

5.16.1.9 diffservSET_DSCLASS_IPv4

```
#define diffservSET_DSCLASS_IPv4(
             pxIPHeader,
             ucValue )
```

Value:

```
    do {                                                                \
        pxIPHeader->ucDifferentiatedServicesCode = ( ucValue « 2 ); \
    } while( ipFALSE_BOOL )
```

5.16.1.10 diffservSET_DSCLASS_IPv6

```
#define diffservSET_DSCLASS_IPv6(
              pxIPHeader,
              ucValue )
```

Value:

```
    do {                                                                    \
        pxIPHeader->ucVersionTrafficClass &= 0xF0;                          \
        pxIPHeader->ucVersionTrafficClass |= ( ( ucValue & 0x3C ) » 2 ); \
        pxIPHeader->ucTrafficClassFlow &= 0x3F;                             \
        pxIPHeader->ucTrafficClassFlow |= ( ( ucValue & 0x3 ) « 6 );     \
    } while( ipFALSE_BOOL )
```

5.16.2 Function Documentation

5.16.2.1 ucDSClassGet()

```
uint8_t ucDSClassGet (
             NetworkBufferDescriptor_t * pxBuf )
```

Retrieves the DiffServ class from the given network buffer.

This function extracts the DiffServ class from the IP header of the network buffer based on the IP version.

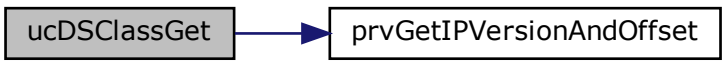
Parameters

in	*pxBuf*	The network buffer descriptor.

Returns

The DiffServ class value.

Here is the call graph for this function:



5.16.2.2 xDSClassSet()

```
BaseType_t xDSClassSet (
             NetworkBufferDescriptor_t * pxBuf,
             uint8_t ucValue )
```

Sets the DiffServ class for the given network buffer.

This function sets the DiffServ class in the IP header of the network buffer based on the IP version.

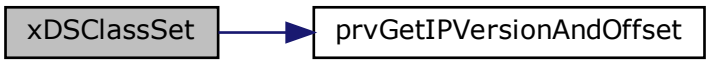
Parameters

in	*pxBuf*	The network buffer descriptor.
in	*ucValue*	The DiffServ class value to set.

Returns

pdPASS if the DiffServ class was set successfully, pdFAIL otherwise.

Here is the call graph for this function:



5.17 FreeRTOS_TSN_DS.h

Go to the documentation of this file.

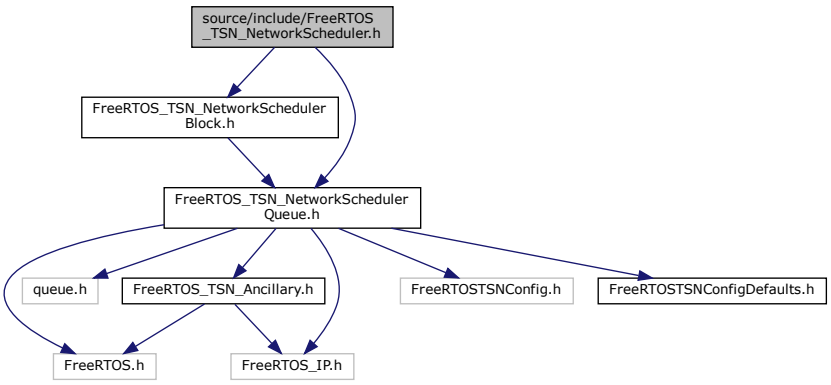
```
1 #ifndef FREERTOS_TSN_DS_H
2 #define FREERTOS_TSN_DS_H
3
4 #include "FreeRTOS.h"
5
6 #include "FreeRTOS_IP.h"
7
8 #define diffservCLASS_DF    ( 0 )
9 /* diffservCLASS_CSx is either 0,8,16,24,32,40,48,54 (CS0 = DF)*/
10 #define diffservCLASS_CSx( x )            ( ( 0 <= x && x <= 7 ) ?  ( 8 * x ) :  diffservCLASS_DF )
11 /* diffservCLASS_AFxy_INET is either 10,12,14,18,20,22,26,28,30,34,36,38*/
12 #define diffservCLASS_AFxy( x, y )        ( ( 1 <= x && x <= 4 && 1 <= y && y <= 3 ) ?  ( 8 * x + 2 * y )
      :  diffservCLASS_DF )
13 #define diffservCLASS_LE    ( 1 )
14 #define diffservCLASS_EF    ( 46 )
15 #define diffservCLASS_DSCP_CUSTOM( x )    ( x & 0x3F )
17 #define diffservGET_DSCLASS_IPv4( pxIPHeader ) \
18 ( pxIPHeader->ucDifferentiatedServicesCode » 2 )
19
20 #define diffservGET_DSCLASS_IPv6( pxIPHeader ) \
21 ( ( ( pxIPHeader->ucVersionTrafficClass & 0xF ) « 2 ) | ( ( pxIPHeader->ucTrafficClassFlow & 0xC0 ) » 6 )
      )
22
23 #define diffservSET_DSCLASS_IPv4( pxIPHeader, ucValue )             \
24 do {                                                                \
25 pxIPHeader->ucDifferentiatedServicesCode = ( ucValue « 2 ); \
26 } while( ipFALSE_BOOL )
27
```

```
28 #define diffservSET_DSCLASS_IPv6( pxIPHeader, ucValue )                  \
29 do {                                                                     \
30 pxIPHeader->ucVersionTrafficClass &= 0xF0;                              \
31 pxIPHeader->ucVersionTrafficClass |= ( ( ucValue & 0x3C ) » 2 ); \
32 pxIPHeader->ucTrafficClassFlow &= 0x3F;                                 \
33 pxIPHeader->ucTrafficClassFlow |= ( ( ucValue & 0x3 ) « 6 );      \
34 } while( ipFALSE_BOOL )
35
36
37 uint8_t ucDSClassGet( NetworkBufferDescriptor_t * pxBuf );
38
39 BaseType_t xDSClassSet( NetworkBufferDescriptor_t * pxBuf,
40                         uint8_t ucValue );
41
42 #endif /* FREERTOS_TSN_DS_H */
```

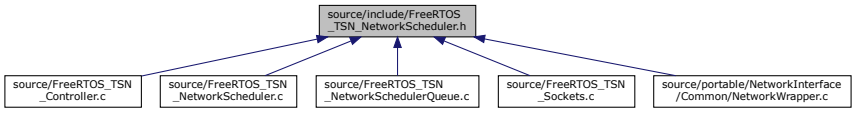
5.18 source/include/FreeRTOS_TSN_NetworkScheduler.h File Reference

```
#include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
#include "FreeRTOS_TSN_NetworkSchedulerQueue.h"
```

Include dependency graph for FreeRTOS_TSN_NetworkScheduler.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xQUEUE_LIST

  A list of network queue pointer.

Typedefs

- typedef struct xQUEUE_LIST NetworkQueueList_t

Functions

- void vNetworkQueueListAdd (NetworkQueueList_t *pxItem)

  Adds a network queue to the network queue list.
- BaseType_t xNetworkQueueAssignRoot (NetworkNode_t *pxNode)

  Assigns the root network node.
- void vNetworkQueueInit (void)
- BaseType_t xNetworkQueueInsertPacketByFilter (const NetworkQueueItem_t *pxItem, UBaseType_t ux↩ Timeout)

  Iterate over the list of network queues and find a match based on the queues' filtering policy. If more than one queue matches the filter, the one with the highest IPV (Internet Protocol Version) is chosen.
- BaseType_t xNetworkQueueInsertPacketByName (const NetworkQueueItem_t *pxItem, char *pcQueue↩ Name, UBaseType_t uxTimeout)

  Inserts a network queue item into a network queue based on the queue name.
- NetworkQueue_t * xNetworkQueueSchedule (void)

  Schedules the network queues and returns the chosen network queue.
- BaseType_t xNetworkQueuePush (NetworkQueue_t *pxQueue, const NetworkQueueItem_t *pxItem, UBaseType_t uxTimeout)

  Pushes a network queue item into a network queue.
- BaseType_t xNetworkQueuePop (NetworkQueue_t *pxQueue, NetworkQueueItem_t *pxItem, UBaseType↩ _t uxTimeout)

  Pops a network queue item from a network queue.
- NetworkQueue_t * pxNetworkQueueFindByName (char *pcName, const NetworkQueueItem_t *pxItem)

5.18.1 Typedef Documentation

5.18.1.1 NetworkQueueList_t

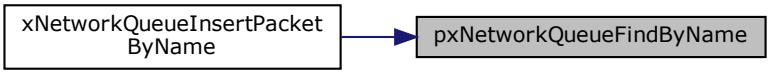
```
typedef struct xQUEUE_LIST NetworkQueueList_t
```

5.18.2 Function Documentation

5.18.2.1 pxNetworkQueueFindByName()

```
NetworkQueue_t * pxNetworkQueueFindByName (
            char * pcName,
            const NetworkQueueItem_t * pxItem )
```

Here is the caller graph for this function:



5.18.2.2 vNetworkQueueInit()

```
void vNetworkQueueInit (
             void  )
```

5.18.2.3 vNetworkQueueListAdd()

```
void vNetworkQueueListAdd (
             NetworkQueueList_t * pxItem )
```

Adds a network queue to the network queue list.

This function is used to add a network queue to the network queue list. The network queue list is a linked list that keeps track of all the network queues.

Parameters

pxItem	The network queue to add.

5.18.2.4 xNetworkQueueAssignRoot()

```
BaseType_t xNetworkQueueAssignRoot (
             NetworkNode_t * pxNode )
```

Assigns the root network node.

This function assigns the specified network node as the root node for the TSN controller's scheduling function. The root node is the starting point for the scheduling algorithm.

Parameters

pxNode	The network node to assign as the root.

Returns

pdPASS if the root network node is successfully assigned, pdFAIL otherwise.

5.18.2.5 xNetworkQueueInsertPacketByFilter()

```
BaseType_t xNetworkQueueInsertPacketByFilter (
             const NetworkQueueItem_t * pxItem,
             UBaseType_t uxTimeout )
```

Iterate over the list of network queues and find a match based on the queues' filtering policy. If more than one queue matches the filter, the one with the highest IPV (Internet Protocol Version) is chosen.

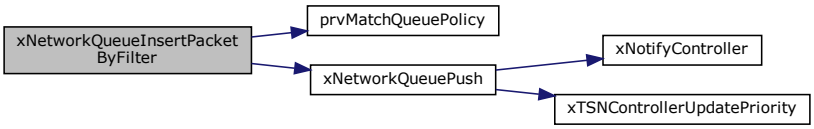
Parameters

pxItem	The network queue item to insert.
uxTimeout	The timeout value for the insertion operation.

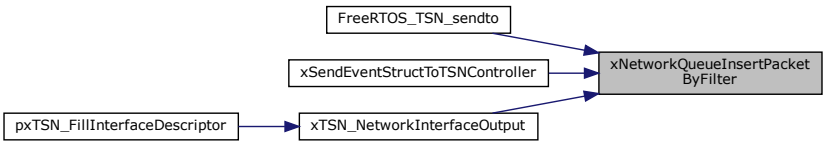
Returns

pdPASS if the network queue item is successfully inserted, pdFAIL otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



5.18.2.6 xNetworkQueueInsertPacketByName()

```
BaseType_t xNetworkQueueInsertPacketByName (
            const NetworkQueueItem_t * pxItem,
            char * pcQueueName,
            UBaseType_t uxTimeout )
```

Inserts a network queue item into a network queue based on the queue name.

This function inserts a network queue item into a network queue identified by its name. The network queue item is inserted using the xNetworkQueuePush() function.

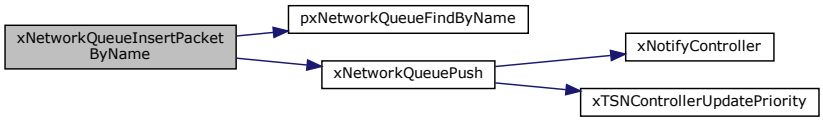
Parameters

pxItem	The network queue item to insert.
pcQueueName	The name of the network queue to insert into.
uxTimeout	The timeout value for the insertion operation.

Returns

pdPASS if the network queue item is successfully inserted, pdFAIL otherwise.

Here is the call graph for this function:



5.18.2.7 xNetworkQueuePop()

```
BaseType_t xNetworkQueuePop (
            NetworkQueue_t * pxQueue,
            NetworkQueueItem_t * pxItem,
            UBaseType_t uxTimeout )
```

Pops a network queue item from a network queue.

This function pops an item from the specified network queue. If the queue is empty, the function will wait for a specified timeout period for an item to become available.

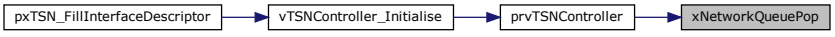
Parameters

pxQueue	The network queue to pop the item from.
pxItem	The network queue item to pop.
uxTimeout	The timeout value for the pop operation.

Returns

pdPASS if a network queue item is successfully popped, pdFAIL otherwise.

Here is the caller graph for this function:



5.18.2.8 xNetworkQueuePush()

```
BaseType_t xNetworkQueuePush (
              NetworkQueue_t * pxQueue,
              const NetworkQueueItem_t * pxItem,
              UBaseType_t uxTimeout )
```

Pushes a network queue item into a network queue.

This function pushes a network queue item into a network queue. It first calls the `fnOnPush` callback function if queue event callbacks are enabled. Then, it uses the `xQueueSendToBack` function to send the item to the back of the queue. If the item is successfully pushed, it updates the priority of the TSN controller (if dynamic priority is enabled), notifies the controller, and returns `pdPASS`. Otherwise, it returns `pdFAIL`.

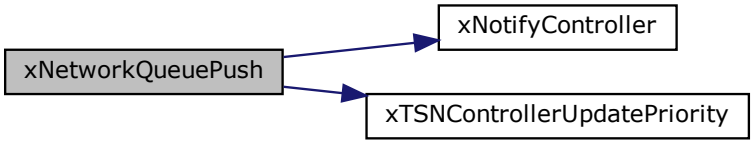
Parameters

pxQueue	The network queue to push the item into.
pxItem	The network queue item to push.
uxTimeout	The timeout value for the push operation.

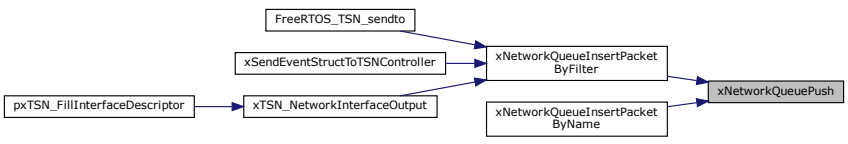
Returns

pdPASS if the network queue item is successfully pushed, pdFAIL otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



5.18.2.9 xNetworkQueueSchedule()

```
NetworkQueue_t * xNetworkQueueSchedule (
            void  )
```

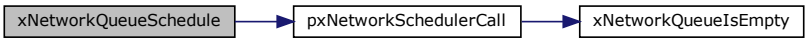
Schedules the network queues and returns the chosen network queue.

This function is responsible for scheduling the network queues and selecting the next network queue to be processed. It checks if there is a network queue available and calls the network scheduler function to make the selection. If there is no network queue available, it returns pdFAIL.

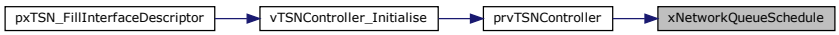
Returns

The chosen network queue, or pdFAIL if no network queue is available.

Here is the call graph for this function:



Here is the caller graph for this function:



5.19 FreeRTOS_TSN_NetworkScheduler.h

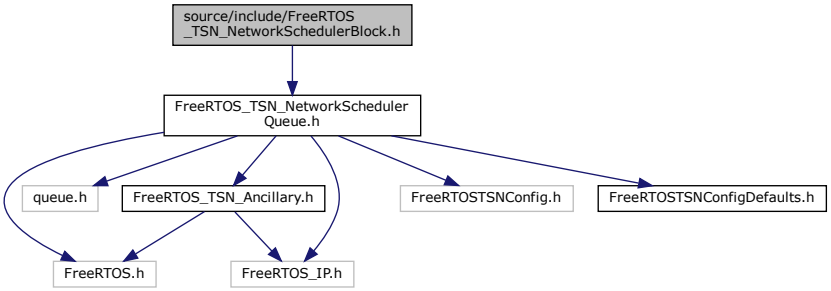
Go to the documentation of this file.

```
1 #ifndef FREERTOS_TSN_NETWORK_SCHEDULER_H
2 #define FREERTOS_TSN_NETWORK_SCHEDULER_H
3
4 #include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
5 #include "FreeRTOS_TSN_NetworkSchedulerQueue.h"
6
12 struct xQUEUE_LIST
13 {
14     struct xNETQUEUE * pxQueue;
15     struct xQUEUE_LIST * pxNext;
16 };
17
18 typedef struct xQUEUE_LIST NetworkQueueList_t;
19
20 void vNetworkQueueListAdd( NetworkQueueList_t * pxItem );
21
22 BaseType_t xNetworkQueueAssignRoot( NetworkNode_t * pxNode );
23
24 /* This must be defined by the user */
25 void vNetworkQueueInit( void );
26
27 BaseType_t xNetworkQueueInsertPacketByFilter( const NetworkQueueItem_t * pxItem,
28                                               UBaseType_t uxTimeout );
29
30 BaseType_t xNetworkQueueInsertPacketByName( const NetworkQueueItem_t * pxItem,
31                                             char * pcQueueName,
32                                             UBaseType_t uxTimeout );
33
```

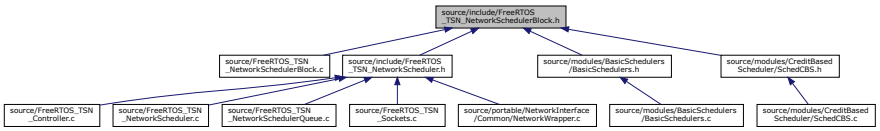
```
34 NetworkQueue_t * xNetworkQueueSchedule( void );
35
36 BaseType_t xNetworkQueuePush( NetworkQueue_t * pxQueue,
37                               const NetworkQueueItem_t * pxItem,
38                               UBaseType_t uxTimeout );
39
40 BaseType_t xNetworkQueuePop( NetworkQueue_t * pxQueue,
41                              NetworkQueueItem_t * pxItem,
42                              UBaseType_t uxTimeout );
43
44 #if ( tsnconfigMAX_QUEUE_NAME_LEN != 0 )
45     NetworkQueue_t * pxNetworkQueueFindByName( char * pcName,
46                                                const NetworkQueueItem_t * pxItem );
47 #endif
48
49 #endif /* FREERTOS_TSN_NETWORK_SCHEDULER_H */
```

5.20 source/include/FreeRTOS_TSN_NetworkSchedulerBlock.h File Reference

`#include "FreeRTOS_TSN_NetworkSchedulerQueue.h"`
Include dependency graph for FreeRTOS_TSN_NetworkSchedulerBlock.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xNETQUEUE_NODE

  This is the structure the stores the nodes of the network scheduler.
- struct xSCHEDULER_GENERIC

  A generic structure for implementing a scheduler.

Macros

- #define netschedCALL_SELECT_FROM_NODE(pxNode) (((struct xSCHEDULER_GENERIC *) pxNode->pvScheduler)->fnSelect(pxNode))
- #define netschedCALL_READY_FROM_NODE(pxNode) (((struct xSCHEDULER_GENERIC *) pxNode->pvScheduler)->fnReady(pxNode))

Typedefs

- typedef struct xNETQUEUE_NODE NetworkNode_t
- typedef NetworkQueue_t *(* SelectQueueFunction_t) (NetworkNode_t *pxNode)
- typedef BaseType_t(* ReadyQueueFunction_t) (NetworkNode_t *pxNode)

Functions

- BaseType_t xNetworkSchedulerLinkQueue (NetworkNode_t *pxNode, NetworkQueue_t *pxQueue)

  Links a network queue to a network node.
- BaseType_t xNetworkSchedulerLinkChild (NetworkNode_t *pxNode, NetworkNode_t *pxChild, size_t uxPosition)

  Links a child network node to a parent network node.
- NetworkQueue_t * pxNetworkSchedulerCall (NetworkNode_t *pxNode)

  Calls the network scheduler for a network node.
- NetworkBufferDescriptor_t * pxPeekNextPacket (NetworkNode_t *pxNode)

  Peeks the next packet in a network node's queue.
- TickType_t uxNetworkQueueGetTicksUntilWakeup (void)

  Gets the ticks until the next wakeup event.
- void vNetworkQueueAddWakeupEvent (TickType_t uxTime)

  Adds a wakeup event to the network queue.

5.20.1 Macro Definition Documentation

5.20.1.1 netschedCALL_READY_FROM_NODE

```
#define netschedCALL_READY_FROM_NODE(
              pxNode )  ( ( ( struct xSCHEDULER_GENERIC * ) pxNode->pvScheduler )->fnReady(
pxNode ) )
```

5.20.1.2 netschedCALL_SELECT_FROM_NODE

```
#define netschedCALL_SELECT_FROM_NODE(
              pxNode )  ( ( ( struct xSCHEDULER_GENERIC * ) pxNode->pvScheduler )->fnSelect(
pxNode ) )
```

5.20.2 Typedef Documentation

5.20.2.1 NetworkNode_t

```
typedef struct xNETQUEUE_NODE NetworkNode_t
```

5.20.2.2 ReadyQueueFunction_t

```
typedef BaseType_t(* ReadyQueueFunction_t) (NetworkNode_t *pxNode)
```

5.20.2.3 SelectQueueFunction_t

```
typedef NetworkQueue_t *(* SelectQueueFunction_t) (NetworkNode_t *pxNode)
```

5.20.3 Function Documentation

5.20.3.1 pxNetworkSchedulerCall()

```
NetworkQueue_t * pxNetworkSchedulerCall (
            NetworkNode_t * pxNode )
```

Calls the network scheduler for a network node.

This function is the core of the network scheduler. It will recursively call the ready function and the select function of the nodes, starting from the root until a ready node is found.

Parameters

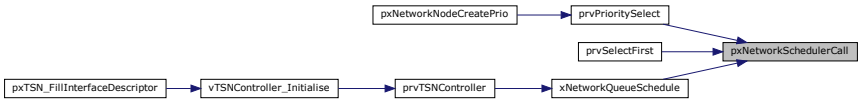
pxNode	Pointer to the network node.

Returns

Pointer to the selected network queue.

Here is the call graph for this function:

Here is the caller graph for this function:



5.20.3.2 pxPeekNextPacket()

```
NetworkBufferDescriptor_t * pxPeekNextPacket (
            NetworkNode_t * pxNode )
```

Peeks the next packet in a network node's queue.

This function is used to peek the next packet in a network node's queue.

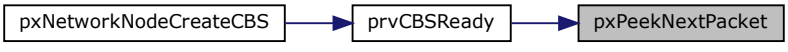
Parameters

pxNode	Pointer to the network node.

Returns

Pointer to the next packet, or NULL if the queue is empty.

Here is the caller graph for this function:



5.20.3.3 uxNetworkQueueGetTicksUntilWakeup()

```
TickType_t uxNetworkQueueGetTicksUntilWakeup (
            void  )
```

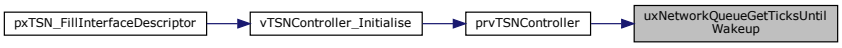
Gets the ticks until the next wakeup event.

This function is used to get the number of ticks until the next wakeup event.

Returns

Number of ticks until the next wakeup event.

Here is the caller graph for this function:



5.20.3.4 vNetworkQueueAddWakeupEvent()

```
void vNetworkQueueAddWakeupEvent (
            TickType_t uxTime )
```

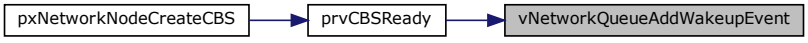
Adds a wakeup event to the network queue.

Altough the network scheduler will always periodicaly checks for new messages, calling this function can help speed up serving waiting packets. Any scheduler that has implemented a ready function that not always returns true should think of suggesting the TSN controller when to check again.

Parameters

uxTime	Time at which to add the wakeup event.

Here is the caller graph for this function:



5.20.3.5 xNetworkSchedulerLinkChild()

```
BaseType_t xNetworkSchedulerLinkChild (
              NetworkNode_t * pxNode,
              NetworkNode_t * pxChild,
              size_t uxPosition )
```

Links a child network node to a parent network node.

This function is used to link a child network node to a parent network node at the specified position.

Parameters

pxNode	Pointer to the parent network node.
pxChild	Pointer to the child network node.
uxPosition	Position at which to link the child network node.

Returns

pdPASS if the link is successful, pdFAIL otherwise.

5.20.3.6 xNetworkSchedulerLinkQueue()

```
BaseType_t xNetworkSchedulerLinkQueue (
              NetworkNode_t * pxNode,
              NetworkQueue_t * pxQueue )
```

Links a network queue to a network node.

This function is used to link a network queue to a network node.

Parameters

pxNode	Pointer to the network node.
pxQueue	Pointer to the network queue.

Returns

pdPASS if the link is successful, pdFAIL otherwise.

5.21 FreeRTOS_TSN_NetworkSchedulerBlock.h

Go to the documentation of this file.

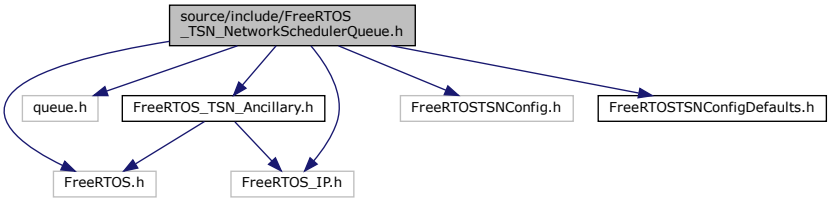
```
1 #ifndef FREERTOS_TSN_NETWORK_SCHEDULER_BLOCK_H
2 #define FREERTOS_TSN_NETWORK_SCHEDULER_BLOCK_H
3
4 #include "FreeRTOS_TSN_NetworkSchedulerQueue.h"
5
13 struct xNETQUEUE_NODE
```

```
14 {
15     uint8_t ucNumChildren;
16     void * pvScheduler;
17     struct xNETQUEUE * pxQueue;
18     struct xNETQUEUE_NODE * pxNext[];
19 };
20
21 typedef struct xNETQUEUE_NODE NetworkNode_t;
22
23 typedef NetworkQueue_t * ( * SelectQueueFunction_t ) ( NetworkNode_t * pxNode );
24
25 typedef BaseType_t ( * ReadyQueueFunction_t ) ( NetworkNode_t * pxNode );
26
39 struct xSCHEDULER_GENERIC
40 {
41     uint16_t usSize;
42     struct xNETQUEUE_NODE * pxOwner;
43     SelectQueueFunction_t fnSelect;
44     ReadyQueueFunction_t fnReady;
45     char ucAttributes[];
46 };
47
48 #if ( configSUPPORT_DYNAMIC_ALLOCATION != 0 )
49
50     NetworkNode_t * pxNetworkNodeCreate( UBaseType_t uxNumChildren );
51
52     void vNetworkNodeRelease( NetworkNode_t * pxNode );
53
54     void * pvNetworkSchedulerGenericCreate( NetworkNode_t * pxNode,
55                                             uint16_t usSize );
56
57     void vNetworkSchedulerGenericRelease( void * pvSched );
58
59 #endif /* if ( configSUPPORT_DYNAMIC_ALLOCATION != 0 ) */
60
61 BaseType_t xNetworkSchedulerLinkQueue( NetworkNode_t * pxNode,
62                                        NetworkQueue_t * pxQueue );
63
64 BaseType_t xNetworkSchedulerLinkChild( NetworkNode_t * pxNode,
65                                        NetworkNode_t * pxChild,
66                                        size_t uxPosition );
67
68 NetworkQueue_t * pxNetworkSchedulerCall( NetworkNode_t * pxNode );
69
70 NetworkBufferDescriptor_t * pxPeekNextPacket( NetworkNode_t * pxNode );
71
72 TickType_t uxNetworkQueueGetTicksUntilWakeup( void );
73
74 void vNetworkQueueAddWakeupEvent( TickType_t uxTime );
75
76 #define netschedCALL_SELECT_FROM_NODE( pxNode ) \
77 ( ( ( struct xSCHEDULER_GENERIC * ) pxNode->pvScheduler )->fnSelect( pxNode ) )
78
79 #define netschedCALL_READY_FROM_NODE( pxNode ) \
80 ( ( ( struct xSCHEDULER_GENERIC * ) pxNode->pvScheduler )->fnReady( pxNode ) )
81
82 #endif /* FREERTOS_TSN_NETWORK_SCHEDULER_H */
```

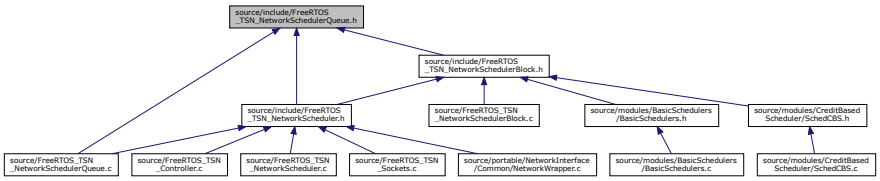
5.22 source/include/FreeRTOS_TSN_NetworkSchedulerQueue.h File Reference

```
#include "FreeRTOS.h"
#include "queue.h"
#include "FreeRTOS_IP.h"
#include "FreeRTOS_TSN_Ancillary.h"
#include "FreeRTOSTSNConfig.h"
#include "FreeRTOSTSNConfigDefaults.h"
```

Include dependency graph for FreeRTOS_TSN_NetworkSchedulerQueue.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xNETQUEUE_ITEM

  The structure used in the network scheduler queues.
- struct xNETQUEUE

  A network queue structure, a leaf in the network scheduler tree.

Typedefs

- typedef BaseType_t(∗ FilterFunction_t) (NetworkBufferDescriptor_t ∗pxNetworkBuffer)
- typedef BaseType_t(∗ PacketHandleFunction_t) (NetworkBufferDescriptor_t ∗pxBuf)
- typedef struct xNETQUEUE_ITEM NetworkQueueItem_t
- typedef struct xNETQUEUE NetworkQueue_t

Enumerations

- enum eQueuePolicy_t { eSendRecv , eSendOnly , eRecvOnly , eIPTaskEvents }

Functions

- UBaseType_t uxNetworkQueuePacketsWaiting (NetworkQueue_t ∗pxQueue)

  Get the number of packets waiting in a network queue.
- BaseType_t xNetworkQueueIsEmpty (NetworkQueue_t ∗pxQueue)

  Check if a network queue is empty.

5.22.1 Typedef Documentation

5.22.1.1 FilterFunction_t

```
typedef BaseType_t(* FilterFunction_t) (NetworkBufferDescriptor_t *pxNetworkBuffer)
```

5.22.1.2 NetworkQueue_t

```
typedef struct xNETQUEUE NetworkQueue_t
```

5.22.1.3 NetworkQueueItem_t

```
typedef struct xNETQUEUE_ITEM NetworkQueueItem_t
```

5.22.1.4 PacketHandleFunction_t

```
typedef BaseType_t(* PacketHandleFunction_t) (NetworkBufferDescriptor_t *pxBuf)
```

5.22.2 Enumeration Type Documentation

5.22.2.1 eQueuePolicy_t

```
enum eQueuePolicy_t
```

Enumerator

eSendRecv	Queue send and receive events
eSendOnly	Only queue transmissions
eRecvOnly	Only queue receptions
eIPTaskEvents	Queue anything and forward to IP task queue

5.22.3 Function Documentation

5.22.3.1 uxNetworkQueuePacketsWaiting()

```
UBaseType_t uxNetworkQueuePacketsWaiting (
            NetworkQueue_t * pxQueue )
```

Get the number of packets waiting in a network queue.

This function returns the number of packets waiting in a network queue.

Parameters

pxQueue	A pointer to the network queue.

Returns

The number of packets waiting in the network queue.

5.22.3.2 xNetworkQueueIsEmpty()

```
BaseType_t xNetworkQueueIsEmpty (
            NetworkQueue_t * pxQueue )
```

Check if a network queue is empty.

This function checks if a network queue is empty by checking if the number of packets waiting in the queue is zero.

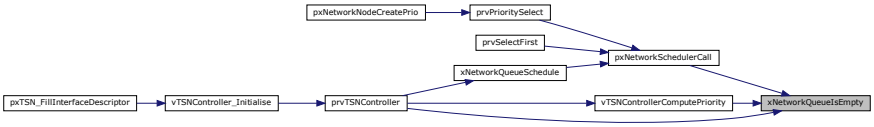
Parameters

pxQueue	A pointer to the network queue.

Returns

pdTRUE if the network queue is empty, pdFALSE otherwise.

Here is the caller graph for this function:



5.23 FreeRTOS_TSN_NetworkSchedulerQueue.h

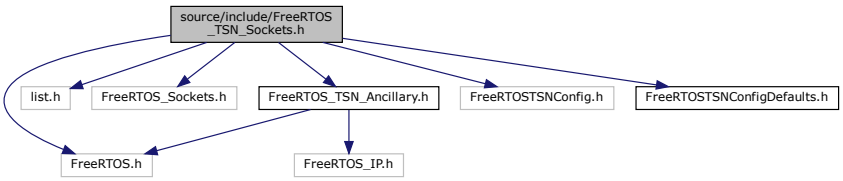
Go to the documentation of this file.

```
1 #ifndef FREERTOS_TSN_NETWORK_SCHEDULER_QUEUE_H
2 #define FREERTOS_TSN_NETWORK_SCHEDULER_QUEUE_H
3
4 #include "FreeRTOS.h"
5 #include "queue.h"
6
7 #include "FreeRTOS_IP.h"
8
9 #include "FreeRTOS_TSN_Ancillary.h"
10
11 #include "FreeRTOSTSNConfig.h"
12 #include "FreeRTOSTSNConfigDefaults.h"
13
14 /* Function pointer to a filtering function.
15 * Used to assign a packet to a network queue.
16 * It should be defined by the user together with queue initialization.
17 * Returns pdTRUE if the packet passes the filter, pdFALSE if not.
18 */
19 typedef BaseType_t ( * FilterFunction_t ) ( NetworkBufferDescriptor_t * pxNetworkBuffer );
20
21 typedef BaseType_t ( * PacketHandleFunction_t ) ( NetworkBufferDescriptor_t * pxBuf );
22
23 typedef enum
24 {
25     eSendRecv,
26     eSendOnly,
27     eRecvOnly,
28     eIPTaskEvents
29 } eQueuePolicy_t;
30
53 struct xNETQUEUE_ITEM
54 {
55     eIPEvent_t eEventType;
56     NetworkBufferDescriptor_t * pxBuf;
57     struct msghdr * pxMsgh;
58     BaseType_t xReleaseAfterSend;
59 };
60
61 typedef struct xNETQUEUE_ITEM NetworkQueueItem_t;
62
87 struct xNETQUEUE
88 {
89     QueueHandle_t xQueue;
90     UBaseType_t uxIPV;
91     eQueuePolicy_t ePolicy;
92 #if ( tsnconfigMAX_QUEUE_NAME_LEN != 0 )
93         char cName[ tsnconfigMAX_QUEUE_NAME_LEN ];
94 #endif
95     FilterFunction_t fnFilter;
96 #if ( tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS != tsnconfigDISABLE )
97         PacketHandleFunction_t fnOnPop;
98         PacketHandleFunction_t fnOnPush;
99 #endif
100 };
101
102 typedef struct xNETQUEUE NetworkQueue_t;
103
104 #if ( configSUPPORT_DYNAMIC_ALLOCATION != 0 )
105
106     NetworkQueue_t * pxNetworkQueueMalloc();
107
108     NetworkQueue_t * pxNetworkQueueCreate( eQueuePolicy_t ePolicy,
109                                            UBaseType_t uxIPV,
110                                            char * cName,
111                                            FilterFunction_t fnFilter );
112
113     void vNetworkQueueFree( NetworkQueue_t * pxQueue );
114
115     NetworkQueueItem_t * pxNetworkQueueItemMalloc();
116
117     void NetworkQueueItemFree( NetworkQueueItem_t * pxItem );
118
119 #endif /* if ( configSUPPORT_DYNAMIC_ALLOCATION != 0 ) */
120
121 UBaseType_t uxNetworkQueuePacketsWaiting( NetworkQueue_t * pxQueue );
122
123 BaseType_t xNetworkQueueIsEmpty( NetworkQueue_t * pxQueue );
124
125 #endif /* FREERTOS_TSN_NETWORK_SCHEDULER_QUEUE_H */
```

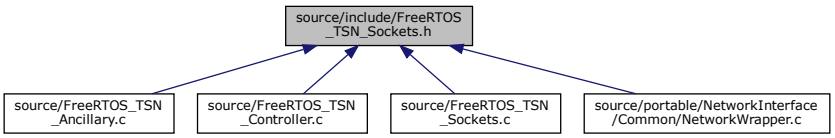
5.24 source/include/FreeRTOS_TSN_Sockets.h File Reference

```
#include "FreeRTOS.h"
#include "list.h"
#include "FreeRTOS_Sockets.h"
#include "FreeRTOS_TSN_Ancillary.h"
#include "FreeRTOSTSNConfig.h"
#include "FreeRTOSTSNConfigDefaults.h"
```

Include dependency graph for FreeRTOS_TSN_Sockets.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xTSN_SOCKET

Macros

- #define FREERTOS_TSN_INVALID_SOCKET ((TSNSocket_t) ~0U)
- #define FREERTOS_SO_DS_CLASS (104)
- #define FREERTOS_SO_TIMESTAMP_OLD (29)
- #define FREERTOS_SO_TIMESTAMPNS_OLD (35)
- #define FREERTOS_SO_TIMESTAMPING_OLD (37)
- #define FREERTOS_SO_TIMESTAMP FREERTOS_SO_TIMESTAMP_OLD
- #define FREERTOS_SO_TIMESTAMPNS FREERTOS_SO_TIMESTAMPNS_OLD
- #define FREERTOS_SO_TIMESTAMPING FREERTOS_SO_TIMESTAMPING_OLD
- #define FREERTOS_SCM_TIMESTAMP FREERTOS_SO_TIMESTAMP
- #define FREERTOS_SCM_TIMESTAMPNS FREERTOS_SO_TIMESTAMPNS
- #define FREERTOS_SCM_TIMESTAMPING FREERTOS_SO_TIMESTAMPING
- #define FREERTOS_SOL_SOCKET (1)
- #define FREERTOS_SOL_IP (0)
- #define FREERTOS_SOL_IPV6 (41)
- #define FREERTOS_IP_RECVERR (11)
- #define FREERTOS_IPV6_RECVERR (25)
- #define FREERTOS_MSG_ERRQUEUE (2 << 13)

Typedefs

- typedef struct xTSN_SOCKET * TSNSocket_t
- typedef struct xTSN_SOCKET FreeRTOS_TSN_Socket_t

Enumerations

- enum {
  SOF_TIMESTAMPING_TX_HARDWARE = (1 << 0) , SOF_TIMESTAMPING_TX_SOFTWARE = (1 << 1) , SOF_TIMESTAMPING_RX_HARDWARE = (1 << 2) , SOF_TIMESTAMPING_RX_SOFTWARE = (1 << 3) ,
  SOF_TIMESTAMPING_SOFTWARE = (1 << 4) , SOF_TIMESTAMPING_SYS_HARDWARE = (1 << 5) , SOF_TIMESTAMPING_RAW_HARDWARE = (1 << 6) , SOF_TIMESTAMPING_OPT_ID = (1 << 7) ,
  SOF_TIMESTAMPING_TX_SCHED = (1 << 8) , SOF_TIMESTAMPING_TX_ACK = (1 << 9) ,
  SOF_TIMESTAMPING_OPT_CMSG = (1 << 10) , SOF_TIMESTAMPING_OPT_TSONLY = (1 << 11) ,
  SOF_TIMESTAMPING_OPT_STATS = (1 << 12) , SOF_TIMESTAMPING_OPT_PKTINFO = (1 << 13) ,
  SOF_TIMESTAMPING_OPT_TX_SWHW = (1 << 14) , SOF_TIMESTAMPING_BIND_PHC = (1 << 15) ,
  SOF_TIMESTAMPING_OPT_ID_TCP = (1 << 16) , SOF_TIMESTAMPING_LAST = SOF_↩ TIMESTAMPING_OPT_ID_TCP , SOF_TIMESTAMPING_MASK }
- enum { SCM_TSTAMP_SND , SCM_TSTAMP_SCHED , SCM_TSTAMP_ACK }

Functions

- void vInitialiseTSNSockets ()
- void vSocketFromPort (TickType_t xSearchKey, Socket_t *pxBaseSocket, TSNSocket_t *pxTSNSocket)

  Searches for a socket based on a given search key and retrieves the corresponding base socket and TSN socket.
- BaseType_t xSocketErrorQueueInsert (TSNSocket_t xTSNSocket, struct msghdr *pxMsgh)
- TSNSocket_t FreeRTOS_TSN_socket (BaseType_t xDomain, BaseType_t xType, BaseType_t xProtocol)

  Creates a TSN socket.
- BaseType_t FreeRTOS_TSN_setsockopt (TSNSocket_t xSocket, int32_t lLevel, int32_t lOptionName, const void *pvOptionValue, size_t uxOptionLength)

  Set socket options for a TSN socket.
- BaseType_t FreeRTOS_TSN_bind (TSNSocket_t xSocket, struct freertos_sockaddr const *pxAddress, socklen_t xAddressLength)

  Binds a TSN socket to a specific address.
- BaseType_t FreeRTOS_TSN_closesocket (TSNSocket_t xSocket)

  Closes a TSN socket.
- int32_t FreeRTOS_TSN_sendto (TSNSocket_t xSocket, const void *pvBuffer, size_t uxTotalDataLength, BaseType_t xFlags, const struct freertos_sockaddr *pxDestinationAddress, socklen_t xDestinationAddress↩ Length)

  Sends data to a TSN socket.
- int32_t FreeRTOS_TSN_recvfrom (TSNSocket_t xSocket, void *pvBuffer, size_t uxBufferLength, BaseType↩ _t xFlags, struct freertos_sockaddr *pxSourceAddress, socklen_t *pxSourceAddressLength)

  Receive data from a TSN socket.
- int32_t FreeRTOS_TSN_recvmsg (TSNSocket_t xSocket, struct msghdr *pxMsghUser, BaseType_t xFlags)

  Receives a message from a TSN socket.

5.24.1 Macro Definition Documentation

5.24.1.1 FREERTOS_IP_RECVERR

```
#define FREERTOS_IP_RECVERR ( 11 )
```

5.24.1.2 FREERTOS_IPV6_RECVERR

```
#define FREERTOS_IPV6_RECVERR ( 25 )
```

5.24.1.3 FREERTOS_MSG_ERRQUEUE

```
#define FREERTOS_MSG_ERRQUEUE ( 2 << 13 )
```

5.24.1.4 FREERTOS_SCM_TIMESTAMP

```
#define FREERTOS_SCM_TIMESTAMP FREERTOS_SO_TIMESTAMP
```

5.24.1.5 FREERTOS_SCM_TIMESTAMPING

```
#define FREERTOS_SCM_TIMESTAMPING FREERTOS_SO_TIMESTAMPING
```

5.24.1.6 FREERTOS_SCM_TIMESTAMPNS

```
#define FREERTOS_SCM_TIMESTAMPNS FREERTOS_SO_TIMESTAMPNS
```

5.24.1.7 FREERTOS_SO_DS_CLASS

```
#define FREERTOS_SO_DS_CLASS ( 104 )
```

5.24.1.8 FREERTOS_SO_TIMESTAMP

```
#define FREERTOS_SO_TIMESTAMP FREERTOS_SO_TIMESTAMP_OLD
```

5.24.1.9 FREERTOS_SO_TIMESTAMP_OLD

```
#define FREERTOS_SO_TIMESTAMP_OLD ( 29 )
```

5.24.1.10 FREERTOS_SO_TIMESTAMPING

```
#define FREERTOS_SO_TIMESTAMPING FREERTOS_SO_TIMESTAMPING_OLD
```

5.24.1.11 FREERTOS_SO_TIMESTAMPING_OLD

```
#define FREERTOS_SO_TIMESTAMPING_OLD ( 37 )
```

5.24.1.12 FREERTOS_SO_TIMESTAMPNS

```
#define FREERTOS_SO_TIMESTAMPNS FREERTOS_SO_TIMESTAMPNS_OLD
```

5.24.1.13 FREERTOS_SO_TIMESTAMPNS_OLD

```
#define FREERTOS_SO_TIMESTAMPNS_OLD ( 35 )
```

5.24.1.14 FREERTOS_SOL_IP

```
#define FREERTOS_SOL_IP ( 0 )
```

5.24.1.15 FREERTOS_SOL_IPV6

```
#define FREERTOS_SOL_IPV6 ( 41 )
```

5.24.1.16 FREERTOS_SOL_SOCKET

```
#define FREERTOS_SOL_SOCKET ( 1 )
```

5.24.1.17 FREERTOS_TSN_INVALID_SOCKET

```
#define FREERTOS_TSN_INVALID_SOCKET ( ( TSNSocket_t ) ~0U )
```

5.24.2 Typedef Documentation

5.24.2.1 FreeRTOS_TSN_Socket_t

```
typedef struct xTSN_SOCKET FreeRTOS_TSN_Socket_t
```

5.24.2.2 TSNSocket_t

```
typedef struct xTSN_SOCKET* TSNSocket_t
```

5.24.3 Enumeration Type Documentation

5.24.3.1 anonymous enum

```
anonymous enum
```

Enumerator

SOF_TIMESTAMPING_TX_HARDWARE	
SOF_TIMESTAMPING_TX_SOFTWARE	
SOF_TIMESTAMPING_RX_HARDWARE	
SOF_TIMESTAMPING_RX_SOFTWARE	
SOF_TIMESTAMPING_SOFTWARE	
SOF_TIMESTAMPING_SYS_HARDWARE	
SOF_TIMESTAMPING_RAW_HARDWARE	
SOF_TIMESTAMPING_OPT_ID	
SOF_TIMESTAMPING_TX_SCHED	
SOF_TIMESTAMPING_TX_ACK	
SOF_TIMESTAMPING_OPT_CMSG	
SOF_TIMESTAMPING_OPT_TSONLY	
SOF_TIMESTAMPING_OPT_STATS	
SOF_TIMESTAMPING_OPT_PKTINFO	
SOF_TIMESTAMPING_OPT_TX_SWHW	
SOF_TIMESTAMPING_BIND_PHC	
SOF_TIMESTAMPING_OPT_ID_TCP	
SOF_TIMESTAMPING_LAST	
SOF_TIMESTAMPING_MASK	

5.24.3.2 anonymous enum

```
anonymous enum
```

Enumerator

SCM_TSTAMP_SND	
SCM_TSTAMP_SCHED	
SCM_TSTAMP_ACK	

5.24.4 Function Documentation

5.24.4.1 FreeRTOS_TSN_bind()

```
BaseType_t FreeRTOS_TSN_bind (
             TSNSocket_t xSocket,
             struct freertos_sockaddr const * pxAddress,
             socklen_t xAddressLength )
```

Binds a TSN socket to a specific address.

This function binds a TSN socket to a specific address specified by `pxAddress`. The `xAddressLength` parameter specifies the length of the address structure.

Parameters

xSocket	The TSN socket to bind.
pxAddress	Pointer to the address structure.
xAddressLength	The length of the address structure.

Returns

If the socket is successfully bound, the function returns 0. Otherwise, it returns a negative value.

5.24.4.2 FreeRTOS_TSN_closesocket()

```
BaseType_t FreeRTOS_TSN_closesocket (
             TSNSocket_t xSocket )
```

Closes a TSN socket.

This function closes the specified TSN socket.

Parameters

xSocket	The TSN socket to be closed.

Returns

If the socket is successfully closed, the function returns 0. Otherwise, it returns an error code.

5.24.4.3 FreeRTOS_TSN_recvfrom()

```
int32_t FreeRTOS_TSN_recvfrom (
            TSNSocket_t xSocket,
            void * pvBuffer,
            size_t uxBufferLength,
            BaseType_t xFlags,
            struct freertos_sockaddr * pxSourceAddress,
            socklen_t * pxSourceAddressLength )
```

Receive data from a TSN socket.

This function receives data from a TSN socket and stores it in the provided buffer.

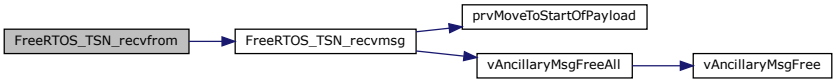
Parameters

xSocket	The TSN socket to receive data from.
pvBuffer	Pointer to the buffer where the received data will be stored.
uxBufferLength	The length of the buffer in bytes.
xFlags	Flags to control the behavior of the receive operation.
pxSourceAddress	Pointer to a structure that will hold the source address information.
pxSourceAddressLength	Pointer to the length of the source address structure.

Returns

The number of bytes received on success, or a negative error code on failure.

Here is the call graph for this function:



5.24.4.4 FreeRTOS_TSN_recvmsg()

```
int32_t FreeRTOS_TSN_recvmsg (
            TSNSocket_t xSocket,
            struct msghdr * pxMsghUser,
            BaseType_t xFlags )
```

Receives a message from a TSN socket.

This function receives a message from the specified TSN socket. It retrieves the message from the waiting packets list of the underlying base socket. If the `FREERTOS_MSG_ERRQUEUE` flag is set, it retrieves the message from the error queue of the TSN socket.

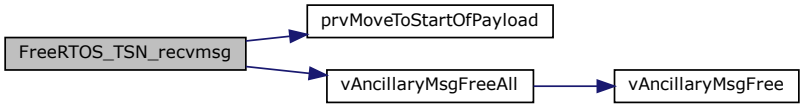
Parameters

xSocket	The TSN socket from which to receive the message.
pxMsghUser	Pointer to the `msghdr` structure that will hold the received message.
xFlags	Flags that control the behavior of the receive operation.

Returns

The length of the payload of the received message, or a negative error code if an error occurs.

Here is the call graph for this function:



Here is the caller graph for this function:

5.24.4.5 FreeRTOS_TSN_sendto()

```
int32_t FreeRTOS_TSN_sendto (
            TSNSocket_t xSocket,
            const void * pvBuffer,
            size_t uxTotalDataLength,
            BaseType_t xFlags,
            const struct freertos_sockaddr * pxDestinationAddress,
            socklen_t xDestinationAddressLength )
```

Sends data to a TSN socket.

This function sends data to a TSN socket specified by the `xSocket` parameter.

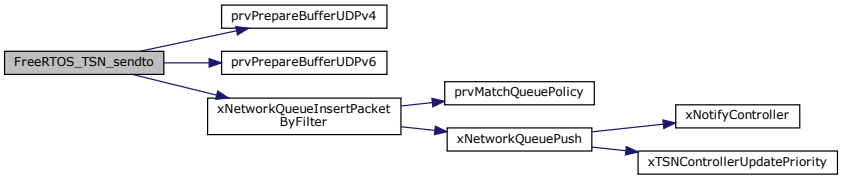
Parameters

xSocket	The TSN socket to send data to.
pvBuffer	Pointer to the data buffer containing the data to send.
uxTotalDataLength	The total length of the data to send.
xFlags	Flags to control the behavior of the send operation.
pxDestinationAddress	Pointer to the destination address structure.
xDestinationAddressLength	The length of the destination address structure.

Returns

The number of bytes sent on success, or a negative error code on failure.

Here is the call graph for this function:



5.24.4.6 FreeRTOS_TSN_setsockopt()

```
BaseType_t FreeRTOS_TSN_setsockopt (
            TSNSocket_t xSocket,
            int32_t lLevel,
            int32_t lOptionName,
            const void * pvOptionValue,
            size_t uxOptionLength )
```

Set socket options for a TSN socket.

This function sets various options for a TSN socket.

Parameters

xSocket	The TSN socket to set options for.
lLevel	The level at which the option is defined.
lOptionName	The name of the option to set.
pvOptionValue	A pointer to the value of the option.
uxOptionLength	The length of the option value.

Returns

pdPASS if the option is set successfully, or a negative value if an error occurs.

5.24.4.7 FreeRTOS_TSN_socket()

```
TSNSocket_t FreeRTOS_TSN_socket (
             BaseType_t xDomain,
             BaseType_t xType,
             BaseType_t xProtocol )
```

Creates a TSN socket.

This function creates a TSN socket with the specified domain, type, and protocol. Only UDP sockets are supported at the moment.

Parameters

xDomain	The domain of the socket.
xType	The type of the socket.
xProtocol	The protocol of the socket.

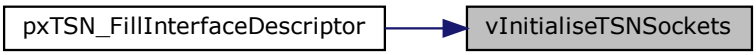
Returns

The created TSN socket, or FREERTOS_TSN_INVALID_SOCKET if an error occurred.

5.24.4.8 vInitialiseTSNSockets()

```
void vInitialiseTSNSockets ( )
```

Here is the caller graph for this function:



5.24.4.9 vSocketFromPort()

```
void vSocketFromPort (
            TickType_t xSearchKey,
            Socket_t * pxBaseSocket,
            TSNSocket_t * pxTSNSocket )
```

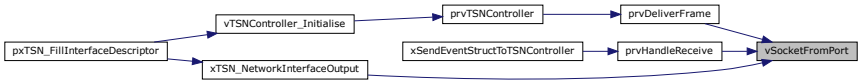
Searches for a socket based on a given search key and retrieves the corresponding base socket and TSN socket.

This function searches for a socket in the TSN bound UDP socket list based on the provided search key. If a matching socket is found, the corresponding TSN socket and base socket are retrieved.

Parameters

xSearchKey	The search key used to find the socket.
pxBaseSocket	Pointer to the base socket variable where the retrieved base socket will be stored.
pxTSNSocket	Pointer to the TSN socket variable where the retrieved TSN socket will be stored.

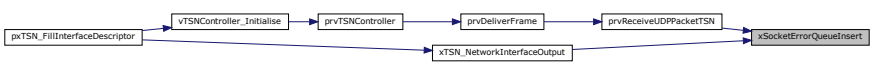
Here is the caller graph for this function:



5.24.4.10 xSocketErrorQueueInsert()

```
BaseType_t xSocketErrorQueueInsert (
            TSNSocket_t xTSNSocket,
            struct msghdr * pxMsgh )
```

Here is the caller graph for this function:



5.25 FreeRTOS_TSN_Sockets.h

Go to the documentation of this file.

```
1 #ifndef FREERTOS_TSN_SOCKETS_H
2 #define FREERTOS_TSN_SOCKETS_H
3
```

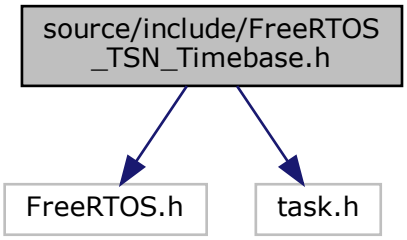
```
4 #include "FreeRTOS.h"
5
6 #include "list.h"
7
8 #include "FreeRTOS_Sockets.h"
9 #include "FreeRTOS_TSN_Ancillary.h"
10
11 #include "FreeRTOSTSNConfig.h"
12 #include "FreeRTOSTSNConfigDefaults.h"
13
14 #define FREERTOS_TSN_INVALID_SOCKET      ( ( TSNSocket_t ) ~0U )
15
16 #if ( tsnconfigSOCKET_INSERTS_VLAN_TAGS != tsnconfigDISABLE ) /*TODO: see SO_PRIORITY */
17 #define FREERTOS_SO_VLAN_CTAG_PCP    ( 101 )
18 #define FREERTOS_SO_VLAN_TAG_PCP     FREERTOS_SO_VLAN_CTAG
19 #define FREERTOS_SO_VLAN_STAG_PCP    ( 102 )
20 #define FREERTOS_SO_VLAN_TAG_RST     ( 103 )
21 #endif
22
23 #define FREERTOS_SO_DS_CLASS             ( 104 )
24
25 #define FREERTOS_SO_TIMESTAMP_OLD        ( 29 )
26 #define FREERTOS_SO_TIMESTAMPNS_OLD      ( 35 )
27 #define FREERTOS_SO_TIMESTAMPING_OLD     ( 37 )
28
29 #define FREERTOS_SO_TIMESTAMP            FREERTOS_SO_TIMESTAMP_OLD
30 #define FREERTOS_SO_TIMESTAMPNS          FREERTOS_SO_TIMESTAMPNS_OLD
31 #define FREERTOS_SO_TIMESTAMPING         FREERTOS_SO_TIMESTAMPING_OLD
32
33 #define FREERTOS_SCM_TIMESTAMP           FREERTOS_SO_TIMESTAMP
34 #define FREERTOS_SCM_TIMESTAMPNS         FREERTOS_SO_TIMESTAMPNS
35 #define FREERTOS_SCM_TIMESTAMPING        FREERTOS_SO_TIMESTAMPING
36
37 #define FREERTOS_SOL_SOCKET              ( 1 )
38 #define FREERTOS_SOL_IP                  ( 0 )
39 #define FREERTOS_SOL_IPV6                ( 41 )
40 #define FREERTOS_IP_RECVERR              ( 11 )
41 #define FREERTOS_IPV6_RECVERR            ( 25 )
42
43 #define FREERTOS_MSG_ERRQUEUE            ( 2 « 13 )
44
45 /* SO_TIMESTAMPING flags */
46 enum
47 {
48     SOF_TIMESTAMPING_TX_HARDWARE = ( 1 « 0 ),
49     SOF_TIMESTAMPING_TX_SOFTWARE = ( 1 « 1 ),
50     SOF_TIMESTAMPING_RX_HARDWARE = ( 1 « 2 ),
51     SOF_TIMESTAMPING_RX_SOFTWARE = ( 1 « 3 ),
52     SOF_TIMESTAMPING_SOFTWARE = ( 1 « 4 ),
53     SOF_TIMESTAMPING_SYS_HARDWARE = ( 1 « 5 ),
54     SOF_TIMESTAMPING_RAW_HARDWARE = ( 1 « 6 ),
55     SOF_TIMESTAMPING_OPT_ID = ( 1 « 7 ),
56     SOF_TIMESTAMPING_TX_SCHED = ( 1 « 8 ),
57     SOF_TIMESTAMPING_TX_ACK = ( 1 « 9 ),
58     SOF_TIMESTAMPING_OPT_CMSG = ( 1 « 10 ),
59     SOF_TIMESTAMPING_OPT_TSONLY = ( 1 « 11 ),
60     SOF_TIMESTAMPING_OPT_STATS = ( 1 « 12 ),
61     SOF_TIMESTAMPING_OPT_PKTINFO = ( 1 « 13 ),
62     SOF_TIMESTAMPING_OPT_TX_SWHW = ( 1 « 14 ),
63     SOF_TIMESTAMPING_BIND_PHC = ( 1 « 15 ),
64     SOF_TIMESTAMPING_OPT_ID_TCP = ( 1 « 16 ),
65
66     SOF_TIMESTAMPING_LAST = SOF_TIMESTAMPING_OPT_ID_TCP,
67     SOF_TIMESTAMPING_MASK = ( SOF_TIMESTAMPING_LAST - 1 ) |
68                             SOF_TIMESTAMPING_LAST
69 };
70
71 /* The type of scm_timestamping, passed in sock_extended_err ee_info.
72 * This defines the type of ts[0].  For SCM_TSTAMP_SND only, if ts[0]
73 * is zero, then this is a hardware timestamp and recorded in ts[2].
74 */
75 enum
76 {
77     SCM_TSTAMP_SND,   /* driver passed skb to NIC, or HW */
78     SCM_TSTAMP_SCHED, /* data entered the packet scheduler */
79     SCM_TSTAMP_ACK,   /* data acknowledged by peer */
80 };
81
82 struct xTSN_SOCKET
83 {
84     Socket_t xBaseSocket;
86     QueueHandle_t xErrQueue;
88 #if ( tsnconfigSOCKET_INSERTS_VLAN_TAGS != tsnconfigDISABLE )
89         uint8_t ucVLANTagsCount;
90         uint16_t usVLANCTagTCI;
91         uint16_t usVLANSTagTCI;
92 #endif
```

```
93     uint8_t ucDSClass;
95     uint32_t ulTSFlags;
97     ListItem_t xBoundSocketListItem;
98     TaskHandle_t xSendTask;
99     TaskHandle_t xRecvTask;
100 };
101
102 typedef struct xTSN_SOCKET   * TSNSocket_t;
103
104 typedef struct xTSN_SOCKET   FreeRTOS_TSN_Socket_t;
105
106 void vInitialiseTSNSockets();
107
108 void vSocketFromPort( TickType_t xSearchKey,
109                       Socket_t * pxBaseSocket,
110                       TSNSocket_t * pxTSNSocket );
111
112 BaseType_t xSocketErrorQueueInsert( TSNSocket_t xTSNSocket,
113                                     struct msghdr * pxMsgh );
114
115 TSNSocket_t FreeRTOS_TSN_socket( BaseType_t xDomain,
116                                  BaseType_t xType,
117                                  BaseType_t xProtocol );
118
119 BaseType_t FreeRTOS_TSN_setsockopt( TSNSocket_t xSocket,
120                                     int32_t lLevel,
121                                     int32_t lOptionName,
122                                     const void * pvOptionValue,
123                                     size_t uxOptionLength );
124
125 BaseType_t FreeRTOS_TSN_bind( TSNSocket_t xSocket,
126                               struct freertos_sockaddr const * pxAddress,
127                               socklen_t xAddressLength );
128
129 BaseType_t FreeRTOS_TSN_closesocket( TSNSocket_t xSocket );
130
131 int32_t FreeRTOS_TSN_sendto( TSNSocket_t xSocket,
132                              const void * pvBuffer,
133                              size_t uxTotalDataLength,
134                              BaseType_t xFlags,
135                              const struct freertos_sockaddr * pxDestinationAddress,
136                              socklen_t xDestinationAddressLength );
137
138 int32_t FreeRTOS_TSN_recvfrom( TSNSocket_t xSocket,
139                                void * pvBuffer,
140                                size_t uxBufferLength,
141                                BaseType_t xFlags,
142                                struct freertos_sockaddr * pxSourceAddress,
143                                socklen_t * pxSourceAddressLength );
144
145 int32_t FreeRTOS_TSN_recvmsg( TSNSocket_t xSocket,
146                               struct msghdr * pxMsghUser,
147                               BaseType_t xFlags );
148
149 #endif /* FREERTOS_TSN_SOCKETS_H */
```

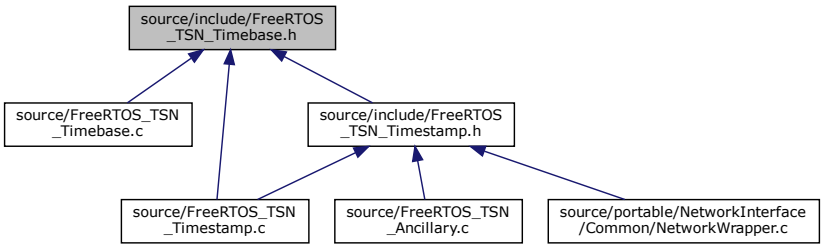
5.26 source/include/FreeRTOS_TSN_Timebase.h File Reference

```
#include "FreeRTOS.h"
#include "task.h"
```

Include dependency graph for FreeRTOS_TSN_Timebase.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct freertos_timespec

  FreeRTOS implementation of a timespec.
- struct xTIMEBASE

Typedefs

- typedef void(∗ TimeBaseStartFunction_t) (void)
- typedef void(∗ TimeBaseStopFunction_t) (void)
- typedef void(∗ TimeBaseSetTimeFunction_t) (const struct freertos_timespec ∗ts)
- typedef void(∗ TimeBaseGetTimeFunction_t) (struct freertos_timespec ∗ts)
- typedef void(∗ TimeBaseAdjTimeFunction_t) (struct freertos_timespec ∗ts, BaseType_t xPositive)
- typedef struct xTIMEBASE TimebaseHandle_t

Enumerations

- enum eTimebaseState_t { eTimebaseNotInitialised = 0 , eTimebaseDisabled , eTimebaseEnabled }

Functions

- void vTimebaseInit (void)

  Initializes and starts the timebase.
- BaseType_t xTimebaseHandleSet (TimebaseHandle_t *pxTimebase)

  Sets the timebase handle.
- void vTimebaseStart (void)

  Starts the timebase.
- void vTimebaseStop (void)
- void vTimebaseSetTime (struct freertos_timespec *ts)

  Sets the time of the timebase.
- void vTimebaseGetTime (struct freertos_timespec *ts)

  Gets the current time of the timebase.
- void vTimebaseAdjTime (struct freertos_timespec *ts, BaseType_t xPositive)
- BaseType_t xTimebaseGetState (void)

  Gets the state of the timebase.
- BaseType_t xTimespecSum (struct freertos_timespec *pxOut, struct freertos_timespec *pxOp1, struct freertos_timespec *pxOp2)

  Sums two timespec structures.
- BaseType_t xTimespecDiff (struct freertos_timespec *pxOut, struct freertos_timespec *pxOp1, struct freertos_timespec *pxOp2)

  Subtracts two timespec structures.
- BaseType_t xTimespecDiv (struct freertos_timespec *pxOut, struct freertos_timespec *pxOp1, BaseType_t xOp2)

  Divides a timespec structure by a scalar value.
- BaseType_t xTimespecCmp (struct freertos_timespec *pxOp1, struct freertos_timespec *pxOp2)

  Compares two timespec structures.

5.26.1 Typedef Documentation

5.26.1.1 TimeBaseAdjTimeFunction_t

```
typedef void(* TimeBaseAdjTimeFunction_t) (struct freertos_timespec *ts, BaseType_t xPositive)
```

5.26.1.2 TimeBaseGetTimeFunction_t

```
typedef void(* TimeBaseGetTimeFunction_t) (struct freertos_timespec *ts)
```

5.26.1.3 TimebaseHandle_t

```
typedef struct xTIMEBASE TimebaseHandle_t
```

5.26.1.4 TimeBaseSetTimeFunction_t

```
typedef void(* TimeBaseSetTimeFunction_t) (const struct freertos_timespec *ts)
```

5.26.1.5 TimeBaseStartFunction_t

```
typedef void(* TimeBaseStartFunction_t) (void)
```

5.26.1.6 TimeBaseStopFunction_t

```
typedef void(* TimeBaseStopFunction_t) (void)
```

5.26.2 Enumeration Type Documentation

5.26.2.1 eTimebaseState_t

```
enum eTimebaseState_t
```

Enumerator

eTimebaseNotInitialised	
eTimebaseDisabled	
eTimebaseEnabled	

5.26.3 Function Documentation

5.26.3.1 vTimebaseAdjTime()

```
void vTimebaseAdjTime (
            struct freertos_timespec * ts,
            BaseType_t xPositive )
```

5.26.3.2 vTimebaseGetTime()

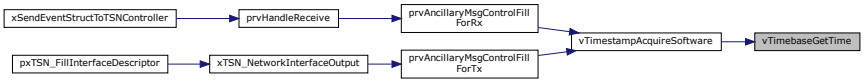
```
void vTimebaseGetTime (
            struct freertos_timespec * ts )
```

Gets the current time of the timebase.

Parameters

ts	Pointer to the timespec structure to store the current time.

Here is the caller graph for this function:



5.26.3.3 vTimebaseInit()

```
void vTimebaseInit (
            void  )
```

Initializes and starts the timebase.

This function should be defined by the user and is project specific. Please remember that this function is also responsible for starting the timer.

5.26.3.4 vTimebaseSetTime()

```
void vTimebaseSetTime (
            struct freertos_timespec * ts )
```

Sets the time of the timebase.

Parameters

ts	Pointer to the timespec structure containing the time to be set.

5.26.3.5 vTimebaseStart()

```
void vTimebaseStart (
            void  )
```

Starts the timebase.

5.26.3.6 vTimebaseStop()

```
void vTimebaseStop (
            void  )
```

5.26.3.7 xTimebaseGetState()

```
BaseType_t xTimebaseGetState (
            void  )
```

Gets the state of the timebase.

Returns

The state of the timebase.

Here is the caller graph for this function:

5.26.3.8 xTimebaseHandleSet()

```
BaseType_t xTimebaseHandleSet (
            TimebaseHandle_t * pxTimebase )
```

Sets the timebase handle.

Parameters

pxTimebase	Pointer to the timebase handle.

Returns

pdPASS if the timebase handle is set successfully, pdFAIL otherwise.

5.26.3.9 xTimespecCmp()

```
BaseType_t xTimespecCmp (
            struct freertos_timespec * pxOp1,
            struct freertos_timespec * pxOp2 )
```

Compares two timespec structures.

Parameters

pxOp1	Pointer to the first timespec structure.
pxOp2	Pointer to the second timespec structure.

Returns

1 if pxOp1 is greater than pxOp2, 0 if they are equal, -1 if pxOp1 is less than pxOp2.

5.26.3.10 xTimespecDiff()

```
BaseType_t xTimespecDiff (
            struct freertos_timespec * pxOut,
            struct freertos_timespec * pxOp1,
            struct freertos_timespec * pxOp2 )
```

Subtracts two timespec structures.

Parameters

pxOut	Pointer to the timespec structure to store the result.
pxOp1	Pointer to the first timespec structure.
pxOp2	Pointer to the second timespec structure.

Returns

pdPASS if the operation is successful, pdFAIL otherwise.

5.26.3.11 xTimespecDiv()

```
BaseType_t xTimespecDiv (
            struct freertos_timespec * pxOut,
            struct freertos_timespec * pxOp1,
            BaseType_t xOp2 )
```

Divides a timespec structure by a scalar value.

Parameters

pxOut	Pointer to the timespec structure to store the result.
pxOp1	Pointer to the timespec structure to be divided.
xOp2	The scalar value to divide by.

Returns

pdPASS if the operation is successful, pdFAIL otherwise.

Here is the call graph for this function:

5.26.3.12 xTimespecSum()

```
BaseType_t xTimespecSum (
            struct freertos_timespec * pxOut,
            struct freertos_timespec * pxOp1,
            struct freertos_timespec * pxOp2 )
```

Sums two timespec structures.

Parameters

pxOut	Pointer to the timespec structure to store the result.
pxOp1	Pointer to the first timespec structure.
pxOp2	Pointer to the second timespec structure.

Returns

pdPASS if the operation is successful, pdFAIL otherwise.

Here is the caller graph for this function:

5.27 FreeRTOS_TSN_Timebase.h

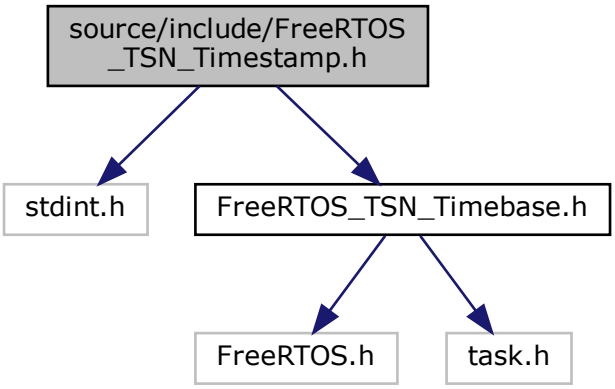
Go to the documentation of this file.

```
1 #ifndef FREERTOS_TSN_TIMEBASE_H
2 #define FREERTOS_TSN_TIMEBASE_H
3
4 #include "FreeRTOS.h"
5 #include "task.h"
6
12 struct freertos_timespec
13 {
14     uint32_t tv_sec;
15     uint32_t tv_nsec;
16 };
17
18 typedef void ( * TimeBaseStartFunction_t )( void );
19
20 typedef void ( * TimeBaseStopFunction_t )( void );
21
22 typedef void ( * TimeBaseSetTimeFunction_t )( const struct freertos_timespec * ts );
23
24 typedef void ( * TimeBaseGetTimeFunction_t )( struct freertos_timespec * ts );
25
26 typedef void ( * TimeBaseAdjTimeFunction_t )( struct freertos_timespec * ts, BaseType_t xPositive );
27
28 typedef enum
29 {
30     eTimebaseNotInitialised = 0,
31     eTimebaseDisabled,
32     eTimebaseEnabled
33 } eTimebaseState_t;
34
35 struct xTIMEBASE
36 {
37     TimeBaseStartFunction_t fnStart;
38     TimeBaseStopFunction_t fnStop;
39     TimeBaseSetTimeFunction_t fnSetTime;
40     TimeBaseGetTimeFunction_t fnGetTime;
41     TimeBaseAdjTimeFunction_t fnAdjTime;
42 };
43
44 typedef struct xTIMEBASE TimebaseHandle_t;
45
53 extern void vTimebaseInit( void );
54
55 BaseType_t xTimebaseHandleSet( TimebaseHandle_t * pxTimebase );
56
57 void vTimebaseStart( void );
58
59 void vTimebaseStop( void );
60
61 void vTimebaseSetTime( struct freertos_timespec * ts );
62
63 void vTimebaseGetTime( struct freertos_timespec * ts );
64
65 void vTimebaseAdjTime( struct freertos_timespec * ts, BaseType_t xPositive );
66
67 BaseType_t xTimebaseGetState( void );
68
69
70 BaseType_t xTimespecSum( struct freertos_timespec * pxOut,
71                          struct freertos_timespec * pxOp1,
72                          struct freertos_timespec * pxOp2 );
73 BaseType_t xTimespecDiff( struct freertos_timespec * pxOut,
74                           struct freertos_timespec * pxOp1,
75                           struct freertos_timespec * pxOp2 );
76 BaseType_t xTimespecDiv( struct freertos_timespec * pxOut,
77                          struct freertos_timespec * pxOp1,
78                          BaseType_t xOp2 );
79 BaseType_t xTimespecCmp( struct freertos_timespec * pxOp1,
80                          struct freertos_timespec * pxOp2 );
81
82
83 #endif /* FREERTOS_TSN_TIMEBASE_H */
```

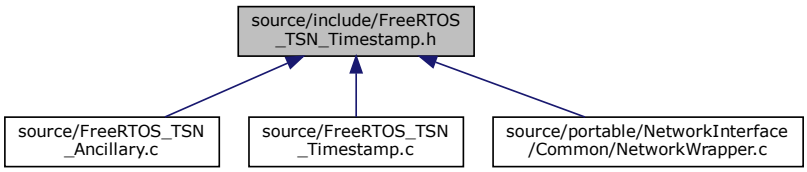
5.28 source/include/FreeRTOS_TSN_Timestamp.h File Reference

```
#include <stdint.h>
#include "FreeRTOS_TSN_Timebase.h"
```

Include dependency graph for FreeRTOS_TSN_Timestamp.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct freertos_scm_timestamping

Functions

- void vTimestampAcquireSoftware (struct freertos_timespec *ts)

  Acquires the software timestamp.

5.28.1 Function Documentation

5.28.1.1 vTimestampAcquireSoftware()

```
void vTimestampAcquireSoftware (
            struct freertos_timespec * ts )
```

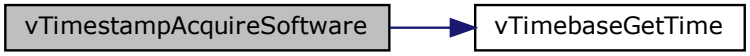
Acquires the software timestamp.

This function acquires the software timestamp by suspending all tasks, getting the time from the timebase, and then resuming all tasks.

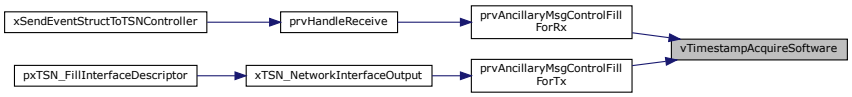
Parameters

ts	Pointer to the freertos_timespec structure where the acquired timestamp will be stored.

Here is the call graph for this function:



Here is the caller graph for this function:



5.29 FreeRTOS_TSN_Timestamp.h

Go to the documentation of this file.

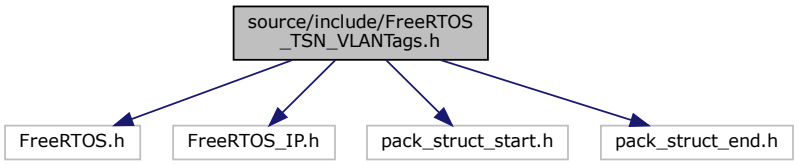
```
1 #ifndef FREERTOS_TSN_TIMESTAMP_H
2 #define FREERTOS_TSN_TIMESTAMP_H
3
4 #include <stdint.h>
5
6 #include "FreeRTOS_TSN_Timebase.h"
7
8 struct freertos_scm_timestamping
9 {
10     struct freertos_timespec ts[ 3 ]; /* ts[0] for software timestamps, ts[2] hw timestamps*/
11 };
12
13 void vTimestampAcquireSoftware( struct freertos_timespec * ts );
14
15 #endif /* FREERTOS_TSN_TIMESTAMP_H */
```

5.30 source/include/FreeRTOS_TSN_VLANTags.h File Reference

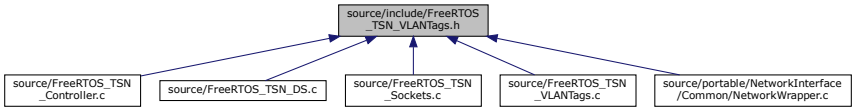
```
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"
#include "pack_struct_start.h"
```

```
#include "pack_struct_end.h"
```

Include dependency graph for FreeRTOS_TSN_VLANTags.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xVLAN_TAG
- struct struct

Macros

- #define vlantagCLASS_0 0U
- #define vlantagCLASS_1 1U
- #define vlantagCLASS_2 2U
- #define vlantagCLASS_3 3U
- #define vlantagCLASS_4 4U
- #define vlantagCLASS_5 5U
- #define vlantagCLASS_6 6U
- #define vlantagCLASS_7 7U
- #define vlantagETH_TAG_OFFSET (12U)
- #define vlantagTPID_DEFAULT (0x8100U)
- #define vlantagTPID_DOUBLE_TAG (0x88a8U)
- #define vlantagPCP_BIT_MASK (0xE000U)
- #define vlantagDEI_BIT_MASK (0x1000U)
- #define vlantagVID_BIT_MASK (0x0FFFU)
- #define vlantagGET_PCP_FROM_TCI(x) ((x & vlantagPCP_BIT_MASK) >> 13)
- #define vlantagGET_DEI_FROM_TCI(x) ((x & vlantagDEI_BIT_MASK) >> 12)
- #define vlantagGET_VID_FROM_TCI(x) ((x & vlantagVID_BIT_MASK))
- #define vlantagSET_PCP_FROM_TCI(x, value)
- #define vlantagSET_DEI_FROM_TCI(x, value)
- #define vlantagSET_VID_FROM_TCI(x, value)
- #define xVLANTagGetPCP xVLANCTagGetPCP
- #define xVLANTagGetDEI xVLANCTagGetDEI
- #define xVLANTagGetVID xVLANCTagGetVID
- #define xVLANTagCheckClass xVLANCTagCheckClass
- #define xVLANTagSetPCP xVLANCTagSetPCP
- #define xVLANTagSetDEI xVLANCTagSetDEI
- #define xVLANTagSetVID xVLANCTagSetVID

Typedefs

- typedef struct xTAGGED_ETH_HEADER TaggedEthernetHeader_t

Functions

- uint8_t ucGetNumberOfTags (NetworkBufferDescriptor_t *pxBuf)

  Get the number of VLAN tags in the given network buffer.
- BaseType_t xVLANSTagGetPCP (NetworkBufferDescriptor_t *pxBuf)

  Get the Priority Code Point (PCP) from the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagGetDEI (NetworkBufferDescriptor_t *pxBuf)

  Get the Drop Eligible Indicator (DEI) from the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagGetVID (NetworkBufferDescriptor_t *pxBuf)

  Get the VLAN Identifier (VID) from the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagCheckClass (NetworkBufferDescriptor_t *pxBuf, BaseType_t xClass)

  Check if the VLAN S-Tag in the network buffer has a specific PCP value.
- BaseType_t xVLANCTagSetPCP (NetworkBufferDescriptor_t *pxBuf, BaseType_t xValue)

  Set the Priority Code Point (PCP) of the VLAN C-Tag in the network buffer.
- BaseType_t xVLANCTagSetDEI (NetworkBufferDescriptor_t *pxBuf, BaseType_t xValue)

  Set the Drop Eligible Indicator (DEI) of the VLAN C-Tag in the network buffer.
- BaseType_t xVLANCTagSetVID (NetworkBufferDescriptor_t *pxBuf, BaseType_t xValue)

  Set the VLAN Identifier (VID) of the VLAN C-Tag in the network buffer.
- BaseType_t xVLANSTagSetPCP (NetworkBufferDescriptor_t *pxBuf, BaseType_t xValue)

  Set the Priority Code Point (PCP) of the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagSetDEI (NetworkBufferDescriptor_t *pxBuf, BaseType_t xValue)

  Set the Drop Eligible Indicator (DEI) of the VLAN S-Tag in the network buffer.
- BaseType_t xVLANSTagSetVID (NetworkBufferDescriptor_t *pxBuf, BaseType_t xValue)

  Set the VLAN Identifier (VID) of the VLAN S-Tag in the network buffer.

Variables

- struct xVLAN_TAG xDestinationAddress
- MACAddress_t xSourceAddress
- struct xVLAN_TAG xVLANTag
- uint16_t usFrameType

5.30.1 Macro Definition Documentation

5.30.1.1 vlantagCLASS_0

```
#define vlantagCLASS_0 0U
```

5.30.1.2 vlantagCLASS_1

```
#define vlantagCLASS_1 1U
```

5.30.1.3 vlantagCLASS_2

```
#define vlantagCLASS_2 2U
```

5.30.1.4 vlantagCLASS_3

```
#define vlantagCLASS_3 3U
```

5.30.1.5 vlantagCLASS_4

```
#define vlantagCLASS_4 4U
```

5.30.1.6 vlantagCLASS_5

```
#define vlantagCLASS_5 5U
```

5.30.1.7 vlantagCLASS_6

```
#define vlantagCLASS_6 6U
```

5.30.1.8 vlantagCLASS_7

```
#define vlantagCLASS_7 7U
```

5.30.1.9 vlantagDEI_BIT_MASK

```
#define vlantagDEI_BIT_MASK ( 0x1000U )
```

5.30.1.10 vlantagETH_TAG_OFFSET

```
#define vlantagETH_TAG_OFFSET ( 12U )
```

5.30.1.11 vlantagGET_DEI_FROM_TCI

```
#define vlantagGET_DEI_FROM_TCI(
              x ) ( ( x & vlantagDEI_BIT_MASK ) >> 12 )
```

5.30.1.12 vlantagGET_PCP_FROM_TCI

```
#define vlantagGET_PCP_FROM_TCI(
              x ) ( ( x & vlantagPCP_BIT_MASK ) >> 13 )
```

5.30.1.13 vlantagGET_VID_FROM_TCI

```
#define vlantagGET_VID_FROM_TCI(
              x ) ( ( x & vlantagVID_BIT_MASK ) )
```

5.30.1.14 vlantagPCP_BIT_MASK

```
#define vlantagPCP_BIT_MASK ( 0xE000U )
```

5.30.1.15 vlantagSET_DEI_FROM_TCI

```
#define vlantagSET_DEI_FROM_TCI(
              x,
              value )
```

Value:

```
    do {                                                                \
        x = ( ( x & ~vlantagDEI_BIT_MASK ) | ( ( value & 0x1U ) << 12 ) ); \
    } while( 0 )
```

5.30.1.16 vlantagSET_PCP_FROM_TCI

```
#define vlantagSET_PCP_FROM_TCI(
              x,
              value )
```

Value:

```
    do {                                                                \
        x = ( ( x & ~vlantagPCP_BIT_MASK ) | ( ( value & 0x7U ) « 13 ) ); \
    } while( 0 )
```

5.30.1.17 vlantagSET_VID_FROM_TCI

```
#define vlantagSET_VID_FROM_TCI(
              x,
              value )
```

Value:

```
    do {                                                          \
        x = ( ( x & ~vlantagVID_BIT_MASK ) | ( ( value & 0xFFFU ) ) ); \
    } while( 0 )
```

5.30.1.18 vlantagTPID_DEFAULT

```
#define vlantagTPID_DEFAULT ( 0x8100U )
```

5.30.1.19 vlantagTPID_DOUBLE_TAG

```
#define vlantagTPID_DOUBLE_TAG ( 0x88a8U )
```

5.30.1.20 vlantagVID_BIT_MASK

```
#define vlantagVID_BIT_MASK ( 0x0FFFU )
```

5.30.1.21 xVLANTagCheckClass

```
#define xVLANTagCheckClass xVLANCTagCheckClass
```

5.30.1.22 xVLANTagGetDEI

```
#define xVLANTagGetDEI xVLANCTagGetDEI
```

5.30.1.23 xVLANTagGetPCP

```
#define xVLANTagGetPCP xVLANCTagGetPCP
```

5.30.1.24 xVLANTagGetVID

```
#define xVLANTagGetVID xVLANCTagGetVID
```

5.30.1.25 xVLANTagSetDEI

```
#define xVLANTagSetDEI xVLANCTagSetDEI
```

5.30.1.26 xVLANTagSetPCP

```
#define xVLANTagSetPCP xVLANCTagSetPCP
```

5.30.1.27 xVLANTagSetVID

```
#define xVLANTagSetVID xVLANCTagSetVID
```

5.30.2 Typedef Documentation

5.30.2.1 TaggedEthernetHeader_t

```
typedef struct xTAGGED_ETH_HEADER TaggedEthernetHeader_t
```

5.30.3 Function Documentation

5.30.3.1 ucGetNumberOfTags()

```
uint8_t ucGetNumberOfTags (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the number of VLAN tags in the given network buffer.

This function checks the Ethernet frame type in the network buffer and determines the number of VLAN tags present.

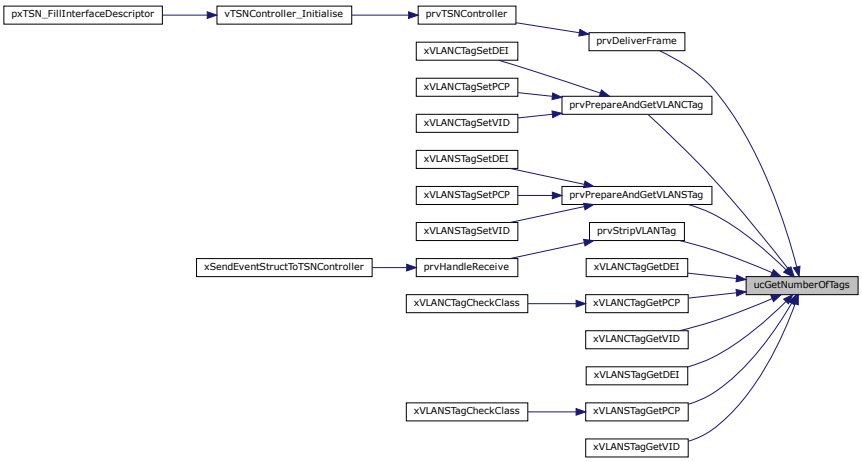
Parameters

pxBuf	Pointer to the network buffer descriptor.

Returns

The number of VLAN tags found in the network buffer.

Here is the caller graph for this function:



5.30.3.2 xVLANCTagSetDEI()

```
BaseType_t xVLANCTagSetDEI (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xValue )
```

Set the Drop Eligible Indicator (DEI) of the VLAN C-Tag in the network buffer.

This function sets the DEI value of the VLAN C-Tag in the network buffer.

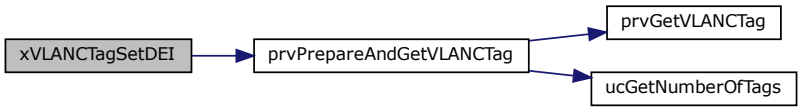
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The DEI value to set.

Returns

pdTRUE if the DEI value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.30.3.3 xVLANCTagSetPCP()

```
BaseType_t xVLANCTagSetPCP (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xValue )
```

Set the Priority Code Point (PCP) of the VLAN C-Tag in the network buffer.

This function sets the PCP value of the VLAN C-Tag in the network buffer.

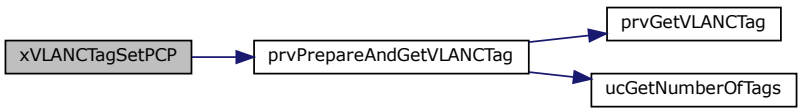
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The PCP value to set.

Returns

pdTRUE if the PCP value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.30.3.4 xVLANCTagSetVID()

```
BaseType_t xVLANCTagSetVID (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xValue )
```

Set the VLAN Identifier (VID) of the VLAN C-Tag in the network buffer.

This function sets the VID value of the VLAN C-Tag in the network buffer.

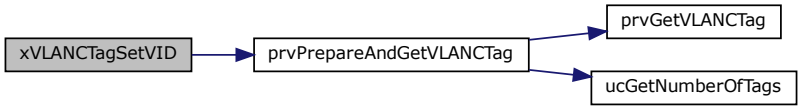
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The VID value to set.

Returns

pdTRUE if the VID value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.30.3.5 xVLANSTagCheckClass()

```
BaseType_t xVLANSTagCheckClass (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xClass )
```

Check if the VLAN S-Tag in the network buffer has a specific PCP value.

This function checks if the PCP value of the VLAN S-Tag in the network buffer matches the specified value.

Parameters

pxBuf	Pointer to the network buffer descriptor.
xClass	The PCP value to check against.

Returns

pdTRUE if the PCP value matches, pdFALSE otherwise.

Here is the call graph for this function:

5.30.3.6 xVLANSTagGetDEI()

```
BaseType_t xVLANSTagGetDEI (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the Drop Eligible Indicator (DEI) from the VLAN S-Tag in the network buffer.

This function retrieves the DEI value from the VLAN S-Tag in the network buffer.

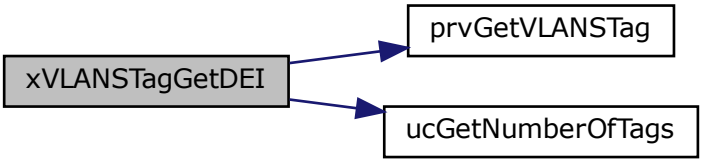
Parameters

pxBuf	Pointer to the network buffer descriptor.

Returns

The DEI value, or ~0 if the VLAN S-Tag is not present.

Here is the call graph for this function:



5.30.3.7 xVLANSTagGetPCP()

```
BaseType_t xVLANSTagGetPCP (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the Priority Code Point (PCP) from the VLAN S-Tag in the network buffer.

This function retrieves the PCP value from the VLAN S-Tag in the network buffer.

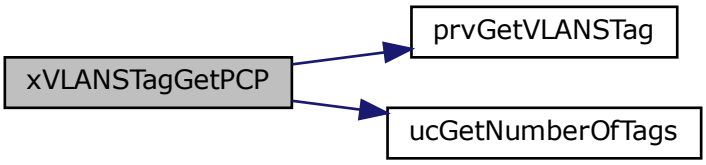
Parameters

pxBuf	Pointer to the network buffer descriptor.

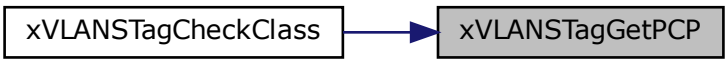
Returns

The PCP value, or ~0 if the VLAN S-Tag is not present.

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.3.8 xVLANSTagGetVID()

```
BaseType_t xVLANSTagGetVID (
            NetworkBufferDescriptor_t * pxBuf )
```

Get the VLAN Identifier (VID) from the VLAN S-Tag in the network buffer.

This function retrieves the VID value from the VLAN S-Tag in the network buffer.

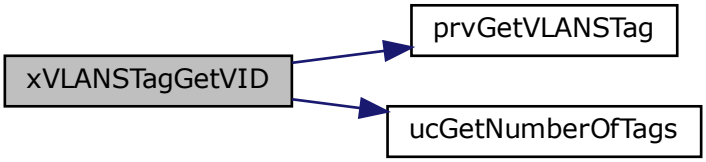
Parameters

pxBuf	Pointer to the network buffer descriptor.

Returns

The VID value, or ~0 if the VLAN S-Tag is not present.

Here is the call graph for this function:



5.30.3.9 xVLANSTagSetDEI()

```
BaseType_t xVLANSTagSetDEI (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xValue )
```

Set the Drop Eligible Indicator (DEI) of the VLAN S-Tag in the network buffer.

This function sets the DEI value of the VLAN S-Tag in the network buffer.

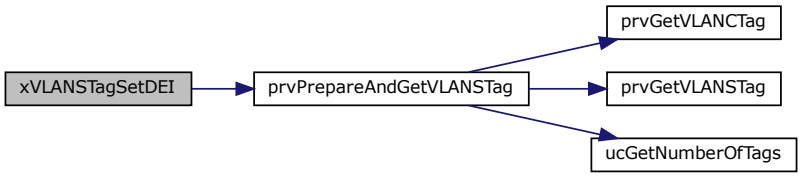
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The DEI value to set.

Returns

pdTRUE if the DEI value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.30.3.10 xVLANSTagSetPCP()

```
BaseType_t xVLANSTagSetPCP (
            NetworkBufferDescriptor_t * pxBuf,
            BaseType_t xValue )
```

Set the Priority Code Point (PCP) of the VLAN S-Tag in the network buffer.

This function sets the PCP value of the VLAN S-Tag in the network buffer.

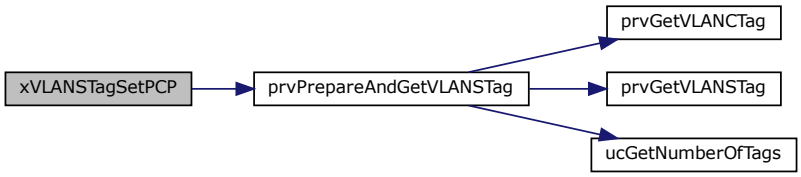
Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The PCP value to set.

Returns

pdTRUE if the PCP value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:



5.30.3.11 xVLANSTagSetVID()

```
BaseType_t xVLANSTagSetVID (
             NetworkBufferDescriptor_t * pxBuf,
             BaseType_t xValue )
```

Set the VLAN Identifier (VID) of the VLAN S-Tag in the network buffer.

This function sets the VID value of the VLAN S-Tag in the network buffer.

Parameters

pxBuf	Pointer to the network buffer descriptor.
xValue	The VID value to set.

Returns

pdTRUE if the VID value is set successfully, pdFALSE otherwise.

Here is the call graph for this function:

5.30.4 Variable Documentation

5.30.4.1 usFrameType

```
uint16_t usFrameType
```

The EtherType field 12 + 2 = 14

5.30.4.2 xDestinationAddress

```
struct xVLAN_TAG xDestinationAddress
```

Destination address 0 + 6 = 6

5.30.4.3 xSourceAddress

```
MACAddress_t xSourceAddress
```

Source address 6 + 6 = 12

5.30.4.4 xVLANTag

```
struct xVLAN_TAG xVLANTag
```

5.31 FreeRTOS_TSN_VLANTags.h

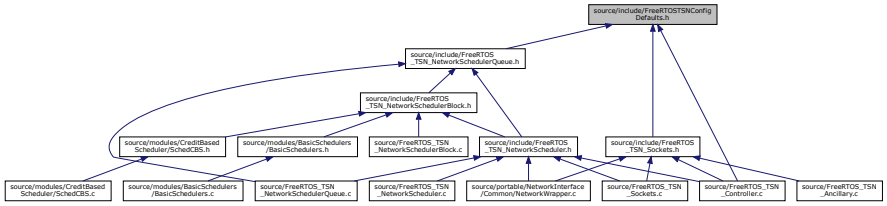
Go to the documentation of this file.

```
1 #ifndef FREERTOS_TSN_VLAN_TAGS_H
2 #define FREERTOS_TSN_VLAN_TAGS_H
3
4 #include "FreeRTOS.h"
5
6 #include "FreeRTOS_IP.h"
7
8 #define vlantagCLASS_0            0U
9 #define vlantagCLASS_1            1U
10 #define vlantagCLASS_2            2U
11 #define vlantagCLASS_3            3U
12 #define vlantagCLASS_4            4U
13 #define vlantagCLASS_5            5U
14 #define vlantagCLASS_6            6U
15 #define vlantagCLASS_7            7U
16
17 #define vlantagETH_TAG_OFFSET     ( 12U )
18
19 #define vlantagTPID_DEFAULT       ( 0x8100U )
20 #define vlantagTPID_DOUBLE_TAG    ( 0x88a8U )
21
22 #define vlantagPCP_BIT_MASK       ( 0xE000U )
23 #define vlantagDEI_BIT_MASK       ( 0x1000U )
24 #define vlantagVID_BIT_MASK       ( 0x0FFFU )
25
26 #define vlantagGET_PCP_FROM_TCI( x )     ( ( x & vlantagPCP_BIT_MASK ) » 13 )
27 #define vlantagGET_DEI_FROM_TCI( x )     ( ( x & vlantagDEI_BIT_MASK ) » 12 )
28 #define vlantagGET_VID_FROM_TCI( x )     ( ( x & vlantagVID_BIT_MASK ) )
29
30 #define vlantagSET_PCP_FROM_TCI( x, value )                                  \
31 do {                                                                        \
32 x = ( ( x & ~vlantagPCP_BIT_MASK ) | ( ( value & 0x7U ) « 13 ) ); \
33 } while( 0 )
34 #define vlantagSET_DEI_FROM_TCI( x, value )                                  \
35 do {                                                                        \
36 x = ( ( x & ~vlantagDEI_BIT_MASK ) | ( ( value & 0x1U ) » 12 ) ); \
37 } while( 0 )
38 #define vlantagSET_VID_FROM_TCI( x, value )                                  \
39 do {                                                                        \
40 x = ( ( x & ~vlantagVID_BIT_MASK ) | ( ( value & 0xFFFU ) ) ); \
41 } while( 0 )
42
43 #include "pack_struct_start.h"
44 struct xVLAN_TAG
45 {
46     uint16_t usTPID;
47     uint16_t usTCI;
48 }
49 #include "pack_struct_end.h"
50
51 #include "pack_struct_start.h"
52 struct xTAGGED_ETH_HEADER
53 {
54     MACAddress_t xDestinationAddress;
55     MACAddress_t xSourceAddress;
56     struct xVLAN_TAG xVLANTag;
57     uint16_t usFrameType;
58 }
59 #include "pack_struct_end.h"
60
61 typedef struct xTAGGED_ETH_HEADER TaggedEthernetHeader_t;
```

```
62
63 #include "pack_struct_start.h"
64 struct xDOUBLE_TAGGED_ETH_HEADER
65 {
66     MACAddress_t xDestinationAddress;
67     MACAddress_t xSourceAddress;
68     struct xVLAN_TAG xVLANSTag;
69     struct xVLAN_TAG xVLANCTag;
70     uint16_t usFrameType;
71 }
72 #include "pack_struct_end.h"
73
74 typedef struct xDOUBLE_TAGGED_ETH_HEADER DoubleTaggedEthernetHeader_t;
75
76 uint8_t ucGetNumberOfTags( NetworkBufferDescriptor_t * pxBuf );
77
78 BaseType_t xVLANSTagGetPCP( NetworkBufferDescriptor_t * pxBuf );
79 BaseType_t xVLANSTagGetDEI( NetworkBufferDescriptor_t * pxBuf );
80 BaseType_t xVLANSTagGetVID( NetworkBufferDescriptor_t * pxBuf );
81 BaseType_t xVLANSTagCheckClass( NetworkBufferDescriptor_t * pxBuf,
82                                 BaseType_t xClass );
83
84 BaseType_t xVLANSTagGetPCP( NetworkBufferDescriptor_t * pxBuf );
85 BaseType_t xVLANSTagGetDEI( NetworkBufferDescriptor_t * pxBuf );
86 BaseType_t xVLANSTagGetVID( NetworkBufferDescriptor_t * pxBuf );
87 BaseType_t xVLANSTagCheckClass( NetworkBufferDescriptor_t * pxBuf,
88                                 BaseType_t xClass );
89
90 /* Defaults to customer tag
91 */
92 #define xVLANTagGetPCP        xVLANCTagGetPCP
93 #define xVLANTagGetDEI        xVLANCTagGetDEI
94 #define xVLANTagGetVID        xVLANCTagGetVID
95 #define xVLANTagCheckClass    xVLANCTagCheckClass
96
97
98 BaseType_t xVLANCTagSetPCP( NetworkBufferDescriptor_t * pxBuf,
99                             BaseType_t xValue );
100 BaseType_t xVLANCTagSetDEI( NetworkBufferDescriptor_t * pxBuf,
101                             BaseType_t xValue );
102 BaseType_t xVLANCTagSetVID( NetworkBufferDescriptor_t * pxBuf,
103                             BaseType_t xValue );
104
105 BaseType_t xVLANSTagSetPCP( NetworkBufferDescriptor_t * pxBuf,
106                             BaseType_t xValue );
107 BaseType_t xVLANSTagSetDEI( NetworkBufferDescriptor_t * pxBuf,
108                             BaseType_t xValue );
109 BaseType_t xVLANSTagSetVID( NetworkBufferDescriptor_t * pxBuf,
110                             BaseType_t xValue );
111
112 #define xVLANTagSetPCP    xVLANCTagSetPCP
113 #define xVLANTagSetDEI    xVLANCTagSetDEI
114 #define xVLANTagSetVID    xVLANCTagSetVID
115
116 #endif /* FREERTOS_TSN_VLAN_TAGS_H */
```

5.32 source/include/FreeRTOSTSNConfigDefaults.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define tsnconfigENABLE (1)

- #define tsnconfigDISABLE (0)
- #define tsnconfigDEFAULT_QUEUE_TIMEOUT (50U)
- #define tsnconfigCONTROLLER_MAX_EVENT_WAIT (1000U)
- #define tsnconfigMAX_QUEUE_NAME_LEN (32U)
- #define tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS tsnconfigDISABLE
- #define tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO tsnconfigDISABLE
- #define tsnconfigTSN_CONTROLLER_PRIORITY (configMAX_PRIORITIES - 1)
- #define tsnconfigWRAPPER_INSERTS_VLAN_TAGS tsnconfigENABLE
- #define tsnconfigSOCKET_INSERTS_VLAN_TAGS tsnconfigDISABLE
- #define tsnconfigERRQUEUE_LENGTH (16)
- #define tsnconfigDUMP_PACKETS tsnconfigDISABLE

5.32.1 Macro Definition Documentation

5.32.1.1 tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO

```
#define tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO tsnconfigDISABLE
```

5.32.1.2 tsnconfigCONTROLLER_MAX_EVENT_WAIT

```
#define tsnconfigCONTROLLER_MAX_EVENT_WAIT ( 1000U )
```

5.32.1.3 tsnconfigDEFAULT_QUEUE_TIMEOUT

```
#define tsnconfigDEFAULT_QUEUE_TIMEOUT ( 50U )
```

5.32.1.4 tsnconfigDISABLE

```
#define tsnconfigDISABLE ( 0 )
```

5.32.1.5 tsnconfigDUMP_PACKETS

```
#define tsnconfigDUMP_PACKETS tsnconfigDISABLE
```

5.32.1.6 tsnconfigENABLE

```
#define tsnconfigENABLE ( 1 )
```

5.32.1.7 tsnconfigERRQUEUE_LENGTH

```
#define tsnconfigERRQUEUE_LENGTH ( 16 )
```

5.32.1.8 tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS

```
#define tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS tsnconfigDISABLE
```

5.32.1.9 tsnconfigMAX_QUEUE_NAME_LEN

```
#define tsnconfigMAX_QUEUE_NAME_LEN ( 32U )
```

5.32.1.10 tsnconfigSOCKET_INSERTS_VLAN_TAGS

```
#define tsnconfigSOCKET_INSERTS_VLAN_TAGS tsnconfigDISABLE
```

5.32.1.11 tsnconfigTSN_CONTROLLER_PRIORITY

```
#define tsnconfigTSN_CONTROLLER_PRIORITY ( configMAX_PRIORITIES - 1 )
```

5.32.1.12 tsnconfigWRAPPER_INSERTS_VLAN_TAGS

```
#define tsnconfigWRAPPER_INSERTS_VLAN_TAGS tsnconfigENABLE
```

5.33 FreeRTOSTSNConfigDefaults.h

Go to the documentation of this file.

```
1 #ifndef FREERTOS_TSN_CONFIG_DEFAULTS_H
2 #define FREERTOS_TSN_CONFIG_DEFAULTS_H
3
4 #ifndef FREERTOS_TSN_CONFIG_H
5 #error FreeRTOSTSNConfig.h has not been included yet
6 #endif
7
8 #define tsnconfigENABLE     ( 1 )
9
10 #define tsnconfigDISABLE    ( 0 )
11
12 /* Queue timeout is used by the TSNController task for timeout on queue
13 * operations.  Note that if any queue is empty the TSN task will never
14 * wait for a message on the single queue, but wait for an event for
15 * tsnconfigCONTROLLER_MAX_EVENT_WAIT ticks.
16 */
17 #ifndef tsnconfigDEFAULT_QUEUE_TIMEOUT
18 #define tsnconfigDEFAULT_QUEUE_TIMEOUT    ( 50U )
19 #endif
20
21 #if ( tsnconfigDEFAULT_QUEUE_TIMEOUT < 0 )
22 #error tsnconfigDEFAULT_QUEUE_TIMEOUT must be a non negative integer
23 #endif
24
25 /* The TSN controller will wake up every time a packet is pushed on its queues
26 * or until the next expected wakeup.  This setting force the controller to
27 * check queues again periodically, so that deadlocks are avoided in case the
28 * user-defined schedulers have any issue.
29 */
30 #ifndef tsnconfigCONTROLLER_MAX_EVENT_WAIT
31 #define tsnconfigCONTROLLER_MAX_EVENT_WAIT    ( 1000U )
32 #endif
33
34 /* The max length for queue names.  This number must take into account also the
35 * space for the string terminator.  A value of 0 means queue names are not used.
36 */
37 #ifndef tsnconfigMAX_QUEUE_NAME_LEN
38 #define tsnconfigMAX_QUEUE_NAME_LEN    ( 32U )
39 #endif
40
41 #if ( tsnconfigMAX_QUEUE_NAME_LEN < 0 )
42 #error tsnconfigMAX_QUEUE_NAME_LEN must be a non negative integer
43 #endif
44
45 /* Enable callbacks when a message is popped and pushed from any network
46 * queue.  The function prototype is defined in FreeRTOS_NetworkSchedulerQueue.h
47 */
48 #ifndef tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS
49 #define tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS    tsnconfigDISABLE
50 #endif
51
52 #if ( ( tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS != tsnconfigDISABLE ) && (
      tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS != tsnconfigENABLE ) )
53 #error Invalid tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS configuration
54 #endif
55
56 /* Used the IPV field in the network queue structure to implement a priority
57 * inheritance mechanism on the network queues.  Note that this is a simplified
58 * version that assumes that whenever a packet is waiting in a queue, a task of
59 * the corresponding priority is always waiting for it.  The TSN controller will
60 * then assume a priority which is the maximum IPV of all the queues with
61 * pending messages.
62 */
63 #ifndef tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO
64 #define tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO    tsnconfigDISABLE
65 #endif
66
67 #if ( ( tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO != tsnconfigDISABLE ) && (
      tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO != tsnconfigENABLE ) )
68 #error Invalid tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO configuration
69 #endif
70
71
72 /* FreeRTOS priority of the TSN controller task.  If tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO
73 * this config entry is ignored as the base priority of the TSN controller is
74 * the priority of the idle task plus 1
75 */
76 #ifndef tsnconfigTSN_CONTROLLER_PRIORITY
77 #define tsnconfigTSN_CONTROLLER_PRIORITY    ( configMAX_PRIORITIES - 1 )
78 #endif
79
80 #if ( ( tsnconfigTSN_CONTROLLER_PRIORITY < 0 ) )
```

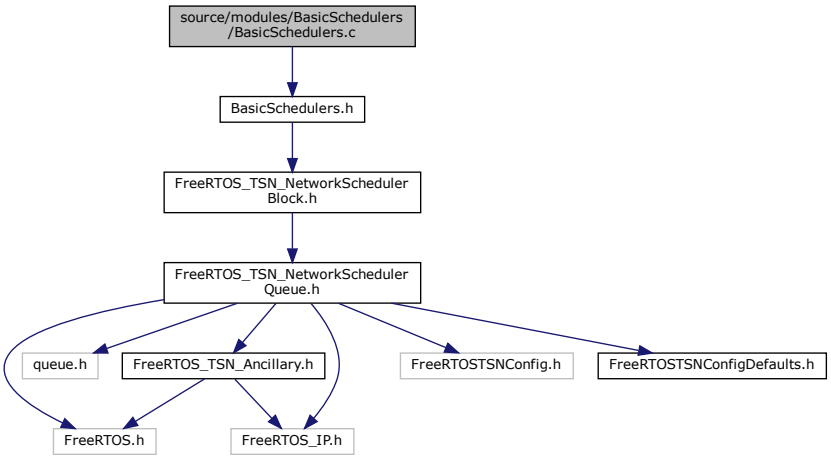
```
81 #error Invalid tsnconfigTSN_CONTROLLER_PRIORITY configuration
82 #endif
83
84 /* If the network interface has no support for adding VLAN tags to 802.1Q
85 * packets, enabling this feature can be a turnaround for sending tagged
86 * packets.  Note that the effect of this option highly depends on the behaviour
87 * of the lower levels (i.e.  some MACs forcefully remove VLAN tags)
88 */
89 #ifndef tsnconfigWRAPPER_INSERTS_VLAN_TAGS
90 #define tsnconfigWRAPPER_INSERTS_VLAN_TAGS    tsnconfigENABLE
91 #endif
92
93 #if ( ( tsnconfigWRAPPER_INSERTS_VLAN_TAGS != tsnconfigDISABLE ) && ( tsnconfigWRAPPER_INSERTS_VLAN_TAGS
      != tsnconfigENABLE ) )
94 #error Invalid tsnconfigWRAPPER_INSERTS_VLAN_TAGS configuration
95 #endif
96
97 /* This option allows the user to set a flag in the sockets to specify the VLAN
98 * tag.  This option reduces compatibility with current +TCP sockets and makes
99 * the API different from Linux sockets.  This is going to be removed in future,
100 * please consider using tsnconfigWRAPPER_INSERTS_VLAN_TAGS instead.
101 */
102 #ifndef tsnconfigSOCKET_INSERTS_VLAN_TAGS
103 #define tsnconfigSOCKET_INSERTS_VLAN_TAGS    tsnconfigDISABLE
104 #endif
105
106 #if ( ( tsnconfigSOCKET_INSERTS_VLAN_TAGS != tsnconfigDISABLE ) && ( tsnconfigSOCKET_INSERTS_VLAN_TAGS
      != tsnconfigENABLE ) )
107 #error Invalid tsnconfigSOCKET_INSERTS_VLAN_TAGS configuration
108 #endif
109
110 #if ( ( tsnconfigWRAPPER_INSERTS_VLAN_TAGS == tsnconfigENABLE ) && ( tsnconfigSOCKET_INSERTS_VLAN_TAGS
      == tsnconfigENABLE ) )
111 #error tsnconfigWRAPPER_INSERTS_VLAN_TAGS and tsnconfigSOCKET_INSERTS_VLAN_TAGS cannot be enabled at the
      same time
112 #endif
113
114 /* The maximum number of messages waiting in a socket errqueue
115 */
116 #ifndef tsnconfigERRQUEUE_LENGTH
117 #define tsnconfigERRQUEUE_LENGTH    ( 16 )
118 #endif
119
120 #if ( tsnconfigERRQUEUE_LENGTH <= 0 )
121 #error Invalid tsnconfigERRQUEUE_LENGTH configuration
122 #endif
123
124 /* Print a dump of ingress/egress packets in hex
125 */
126 #ifndef tsnconfigDUMP_PACKETS
127 #define tsnconfigDUMP_PACKETS    tsnconfigDISABLE
128 #endif
129
130 #if ( ( tsnconfigDUMP_PACKETS != tsnconfigDISABLE ) && ( tsnconfigDUMP_PACKETS != tsnconfigENABLE ) )
131 #error Invalid tsnconfigDUMP_PACKETS configuration
132 #endif
133
134 #endif /* FREERTOS_TSN_CONFIG_DEFAULTS_H */
```

5.34 source/modules/BasicSchedulers/BasicSchedulers.c File Reference

Implementation of some basic schedulers.

```
#include "BasicSchedulers.h"
```

Include dependency graph for BasicSchedulers.c:



Data Structures

- struct xSCHEDULER_RR
- struct xSCHEDULER_PRIO
- struct xSCHEDULER_FIFO

Functions

- NetworkQueue_t * prvPrioritySelect (NetworkNode_t *pxNode)
- NetworkNode_t * pxNetworkNodeCreatePrio (BaseType_t uxNumChildren)
- NetworkNode_t * pxNetworkNodeCreateFIFO ()

5.34.1 Detailed Description

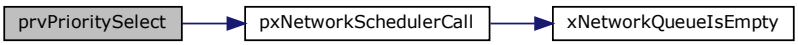
Implementation of some basic schedulers.

5.34.2 Function Documentation

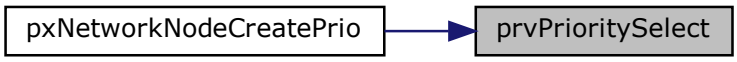
5.34.2.1 prvPrioritySelect()

```
NetworkQueue_t * prvPrioritySelect (
            NetworkNode_t * pxNode )
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.34.2.2 pxNetworkNodeCreateFIFO()

```
NetworkNode_t * pxNetworkNodeCreateFIFO (
            void  )
```

5.34.2.3 pxNetworkNodeCreatePrio()

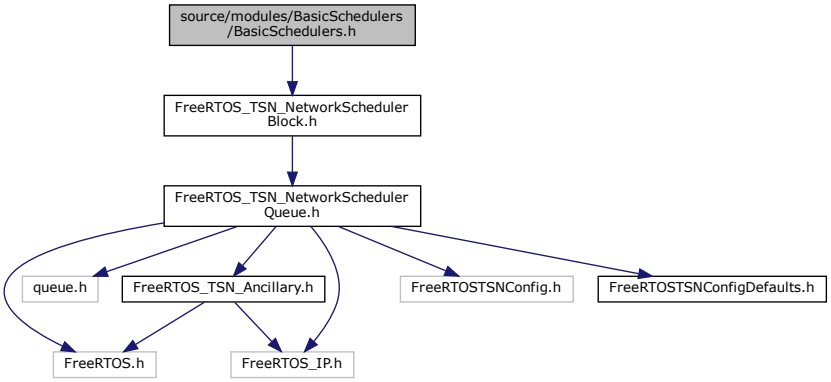
```
NetworkNode_t * pxNetworkNodeCreatePrio (
            BaseType_t uxNumChildren )
```

Here is the call graph for this function:

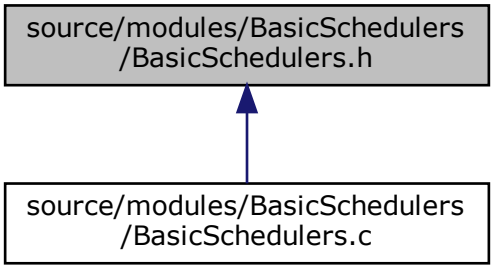
5.35 source/modules/BasicSchedulers/BasicSchedulers.h File Reference

```
#include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
```

Include dependency graph for BasicSchedulers.h:



This graph shows which files directly or indirectly include this file:



Functions

- NetworkNode_t * pxNetworkNodeCreateFIFO (void)
- NetworkNode_t * pxNetworkNodeCreateRR (BaseType_t uxNumChildren)
- NetworkNode_t * pxNetworkNodeCreatePrio (BaseType_t uxNumChildren)

5.35.1 Function Documentation

5.35.1.1 pxNetworkNodeCreateFIFO()

```
NetworkNode_t * pxNetworkNodeCreateFIFO (
            void  )
```

5.35.1.2 pxNetworkNodeCreatePrio()

```
NetworkNode_t * pxNetworkNodeCreatePrio (
            BaseType_t uxNumChildren )
```

Here is the call graph for this function:

5.35.1.3 pxNetworkNodeCreateRR()

```
NetworkNode_t * pxNetworkNodeCreateRR (
            BaseType_t uxNumChildren )
```

5.36 BasicSchedulers.h

Go to the documentation of this file.

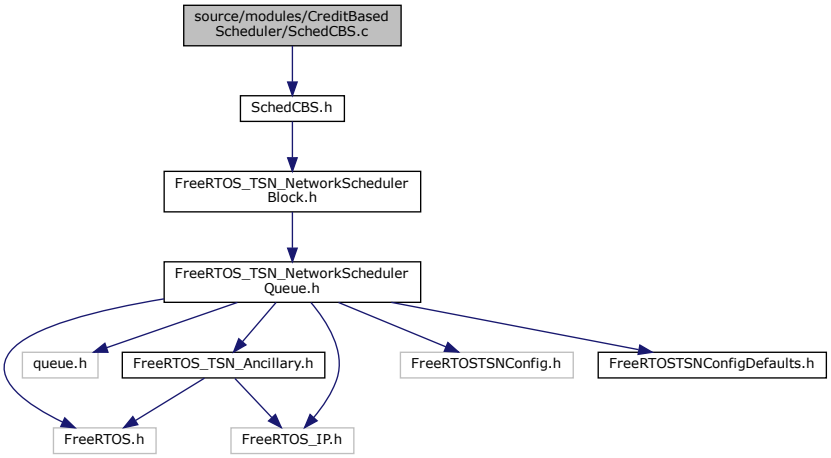
```
1 #ifndef BASIC_SCHEDULERS_H
2 #define BASIC_SCHEDULERS_H
3
4 #include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
5
6 NetworkNode_t * pxNetworkNodeCreateFIFO( void );
7
8 NetworkNode_t * pxNetworkNodeCreateRR( BaseType_t uxNumChildren );
9
10 NetworkNode_t * pxNetworkNodeCreatePrio( BaseType_t uxNumChildren );
11
12 #endif /* BASIC_SCHEDULERS_H */
```

5.37 source/modules/CreditBasedScheduler/SchedCBS.c File Reference

Implementation of a Credit Based Scheduler.

```
#include "SchedCBS.h"
```

Include dependency graph for SchedCBS.c:



Data Structures

- struct xSCHEDULER_CBS

Functions

- BaseType_t prvCBSReady (NetworkNode_t *pxNode)
- NetworkNode_t * pxNetworkNodeCreateCBS (UBaseType_t uxBandwidth, UBaseType_t uxMaxCredit)

  Creates a CBS scheduler.

5.37.1 Detailed Description

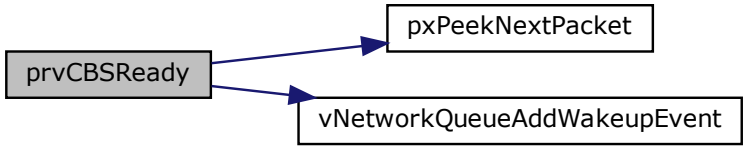
Implementation of a Credit Based Scheduler.

5.37.2 Function Documentation

5.37.2.1 prvCBSReady()

```
BaseType_t prvCBSReady (
            NetworkNode_t * pxNode )
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.37.2.2 pxNetworkNodeCreateCBS()

```
NetworkNode_t * pxNetworkNodeCreateCBS (
            UBaseType_t uxBandwidth,
            UBaseType_t uxMaxCredit )
```

Creates a CBS scheduler.

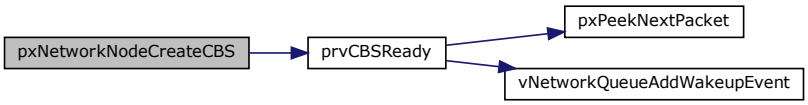
Parameters

uxBandwidth	The desired bandwidth of the scheduler, measured in bit per second
uxMaxCredit	The max credit of the scheduler, in bits

Returns

A pointer to the node with the scheduler

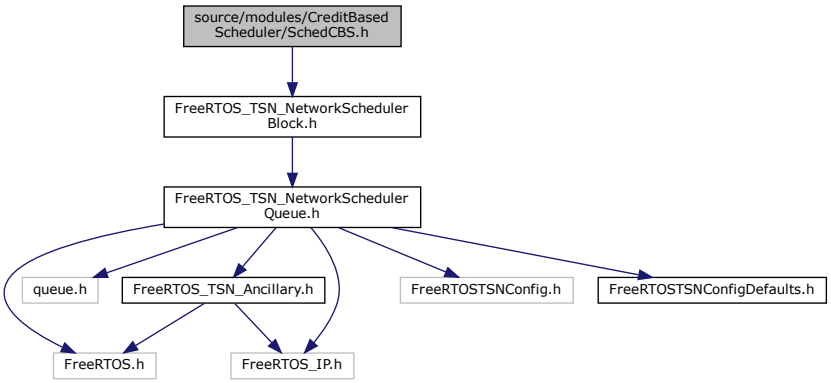
Here is the call graph for this function:



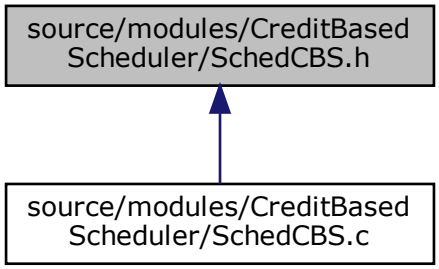
5.38 source/modules/CreditBasedScheduler/SchedCBS.h File Reference

```
#include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
```

Include dependency graph for SchedCBS.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define netschedCBS_DEFAULT_BANDWIDTH (1 << 20)
- #define netschedCBS_DEFAULT_MAXCREDIT (1536 * 2) /* max burst = 2 frames */

Functions

- NetworkNode_t * pxNetworkNodeCreateCBS (UBaseType_t uxBandwidth, UBaseType_t uxMaxCredit)

  Creates a CBS scheduler.

5.38.1 Macro Definition Documentation

5.38.1.1 netschedCBS_DEFAULT_BANDWIDTH

```
#define netschedCBS_DEFAULT_BANDWIDTH ( 1 << 20 )
```

5.38.1.2 netschedCBS_DEFAULT_MAXCREDIT

```
#define netschedCBS_DEFAULT_MAXCREDIT ( 1536 * 2 ) /* max burst = 2 frames */
```

5.38.2 Function Documentation

5.38.2.1 pxNetworkNodeCreateCBS()

```
NetworkNode_t * pxNetworkNodeCreateCBS (
            UBaseType_t uxBandwidth,
            UBaseType_t uxMaxCredit )
```

Creates a CBS scheduler.

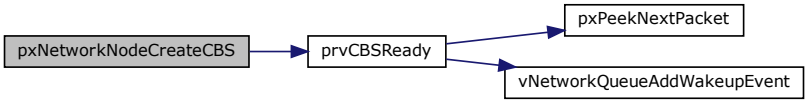
Parameters

uxBandwidth	The desired bandwidth of the scheduler, measured in bit per second
uxMaxCredit	The max credit of the scheduler, in bits

Returns

A pointer to the node with the scheduler

Here is the call graph for this function:



5.39 SchedCBS.h

Go to the documentation of this file.

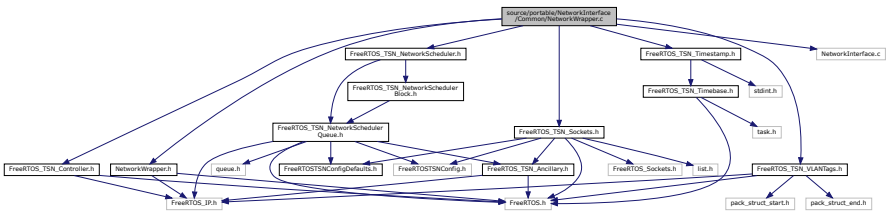
```
1 #ifndef SCHED_CBS_H
2 #define SCHED_CBS_H
3
4 #include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
5
6 #define netschedCBS_DEFAULT_BANDWIDTH    ( 1 << 20 )
7 #define netschedCBS_DEFAULT_MAXCREDIT    ( 1536 * 2 ) /* max burst = 2 frames */
8
9 NetworkNode_t * pxNetworkNodeCreateCBS( UBaseType_t uxBandwidth,
10                                         UBaseType_t uxMaxCredit );
11
12 #endif /* ifndef SCHED_CBS_H */
```

5.40 source/portable/NetworkInterface/Common/NetworkWrapper.c File Reference

A wrapper for the original NetworkInterface.c.

```
#include "NetworkWrapper.h"
#include "FreeRTOS_TSN_Controller.h"
#include "FreeRTOS_TSN_NetworkScheduler.h"
#include "FreeRTOS_TSN_Sockets.h"
#include "FreeRTOS_TSN_VLANTags.h"
#include "FreeRTOS_TSN_Timestamp.h"
#include "NetworkInterface.c"
```

Include dependency graph for NetworkWrapper.c:



Macros

- #define xNetworkInterfaceInitialise xMAC_NetworkInterfaceInitialise
- #define xNetworkInterfaceOutput xMAC_NetworkInterfaceOutput
- #define xGetPhyLinkStatus xMAC_GetPhyLinkStatus
- #define pxFillInterfaceDescriptor pxMAC_FillInterfaceDescriptor
- #define xSendEventStructToIPTask xSendEventStructToTSNController
- #define wrapperFIRST_TPID (0x88a8)
- #define wrapperSECOND_TPID (0x8100)

Functions

- BaseType_t xSendEventStructToTSNController (const IPStackEvent_t *pxEvent, TickType_t uxTimeout)

  Function to send an event structure to the TSN Controller.
- void vNetworkQueueInit ()

  Initializes the network queues and network nodes.
- void vTimebaseInit ()

  Initializes and starts the timebase.
- NetworkBufferDescriptor_t * prvInsertVLANTag (NetworkBufferDescriptor_t *const pxBuf, uint16_t usTCI, uint16_t usTPID)
- NetworkQueueItem_t * prvHandleReceive (NetworkBufferDescriptor_t *pxBuf)
- BaseType_t prvStripVLANTag (NetworkBufferDescriptor_t *pxBuf, uint16_t *pusVLANTCI, uint16_t *pusVLANServiceTCI)
- void prvDumpPacket (char *const pcPrefix, NetworkBufferDescriptor_t *pxBuf)
- BaseType_t prvAncillaryMsgControlFillForRx (struct msghdr *pxMsgh, NetworkBufferDescriptor_t *pxBuf, Socket_t xSocket, TSNSocket_t xTSNSocket)
- BaseType_t prvAncillaryMsgControlFillForTx (struct msghdr *pxMsgh, NetworkBufferDescriptor_t *pxBuf, Socket_t xSocket, TSNSocket_t xTSNSocket)
- BaseType_t xTSN_NetworkInterfaceInitialise (NetworkInterface_t *pxInterface)
- BaseType_t xTSN_NetworkInterfaceOutput (NetworkInterface_t *pxInterface, NetworkBufferDescriptor_t *const pxBuffer, BaseType_t bReleaseAfterSend)

  The function used to send a packet.
- NetworkInterface_t * pxTSN_FillInterfaceDescriptor (BaseType_t xEMACIndex, NetworkInterface_t *pxInterface, NetworkInterfaceConfig_t *pxInterfaceConfig)
- BaseType_t xTSN_GetPhyLinkStatus (NetworkInterface_t *pxInterface)
- BaseType_t xNetworkInterfaceInitialise (NetworkInterface_t *pxInterface)
- BaseType_t xNetworkInterfaceOutput (NetworkInterface_t *pxInterface, NetworkBufferDescriptor_t *const pxBuffer, BaseType_t bReleaseAfterSend)
- BaseType_t xGetPhyLinkStatus (NetworkInterface_t *pxInterface)
- void vRetrieveHardwareTimestamp (NetworkInterface_t *pxInterface, NetworkBufferDescriptor_t *pxBuf, uint32_t *pusSec, uint32_t *pusNanosec)

Variables

- BaseType_t xNetworkWrapperInitialised = pdFALSE

5.40.1 Detailed Description

A wrapper for the original NetworkInterface.c.

This file is a wrapper for the original network interface, it hijacks the calls to the Plus TCP functions to the functions in this library.

5.40.2 Macro Definition Documentation

5.40.2.1 pxFillInterfaceDescriptor

```
#define pxFillInterfaceDescriptor pxMAC_FillInterfaceDescriptor
```

5.40.2.2 wrapperFIRST_TPID

```
#define wrapperFIRST_TPID ( 0x88a8 )
```

5.40.2.3 wrapperSECOND_TPID

```
#define wrapperSECOND_TPID ( 0x8100 )
```

5.40.2.4 xGetPhyLinkStatus

```
#define xGetPhyLinkStatus xMAC_GetPhyLinkStatus
```

5.40.2.5 xNetworkInterfaceInitialise

```
#define xNetworkInterfaceInitialise xMAC_NetworkInterfaceInitialise
```

5.40.2.6 xNetworkInterfaceOutput

```
#define xNetworkInterfaceOutput xMAC_NetworkInterfaceOutput
```

5.40.2.7 xSendEventStructToIPTask

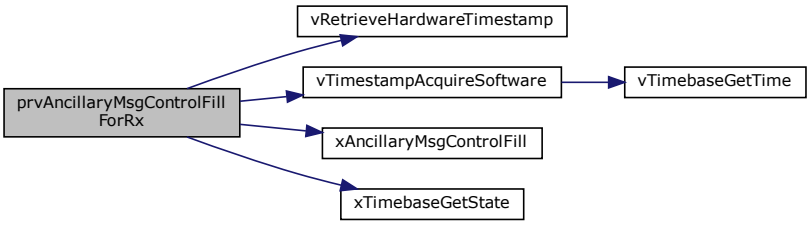
```
#define xSendEventStructToIPTask xSendEventStructToTSNController
```

5.40.3 Function Documentation

5.40.3.1 prvAncillaryMsgControlFillForRx()

```
BaseType_t prvAncillaryMsgControlFillForRx (
            struct msghdr * pxMsgh,
            NetworkBufferDescriptor_t * pxBuf,
            Socket_t xSocket,
            TSNSocket_t xTSNSocket )
```

Here is the call graph for this function:

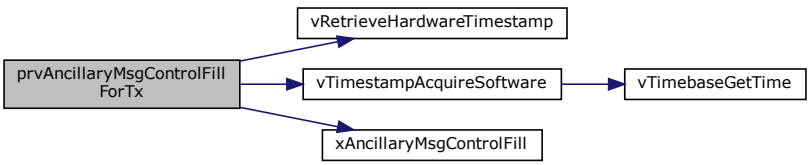


Here is the caller graph for this function:

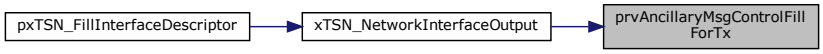
5.40.3.2 prvAncillaryMsgControlFillForTx()

```
BaseType_t prvAncillaryMsgControlFillForTx (
            struct msghdr * pxMsgh,
            NetworkBufferDescriptor_t * pxBuf,
            Socket_t xSocket,
            TSNSocket_t xTSNSocket )
```

Here is the call graph for this function:



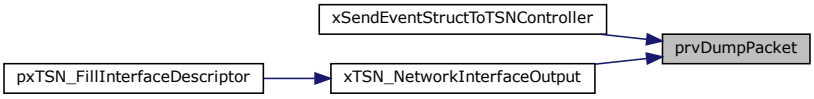
Here is the caller graph for this function:



5.40.3.3 prvDumpPacket()

```
void prvDumpPacket (
            char *const pcPrefix,
            NetworkBufferDescriptor_t * pxBuf )
```

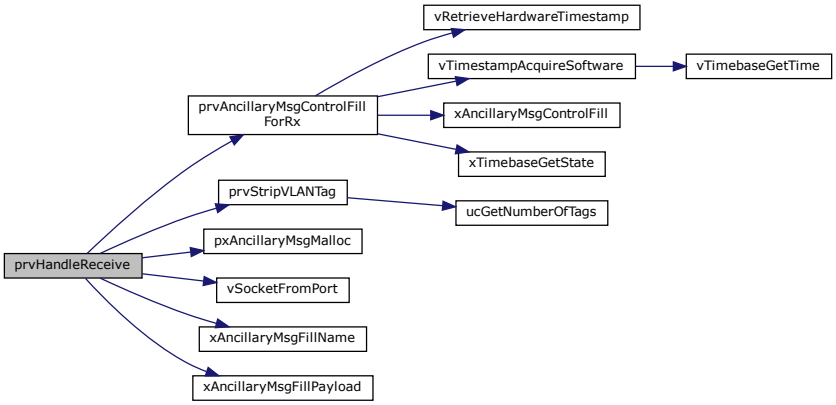
Here is the caller graph for this function:



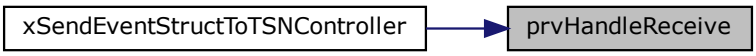
5.40.3.4 prvHandleReceive()

```
NetworkQueueItem_t * prvHandleReceive (
            NetworkBufferDescriptor_t * pxBuf )
```

Here is the call graph for this function:



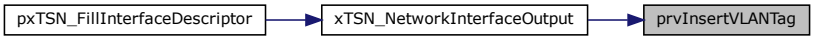
Here is the caller graph for this function:



5.40.3.5 prvInsertVLANTag()

```
NetworkBufferDescriptor_t * prvInsertVLANTag (
            NetworkBufferDescriptor_t *const pxBuf,
            uint16_t usTCI,
            uint16_t usTPID )
```

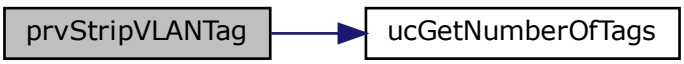
Here is the caller graph for this function:



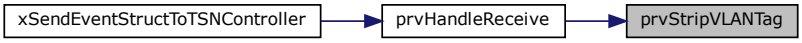
5.40.3.6 prvStripVLANTag()

```
BaseType_t prvStripVLANTag (
            NetworkBufferDescriptor_t * pxBuf,
            uint16_t * pusVLANTCI,
            uint16_t * pusVLANServiceTCI )
```

Here is the call graph for this function:



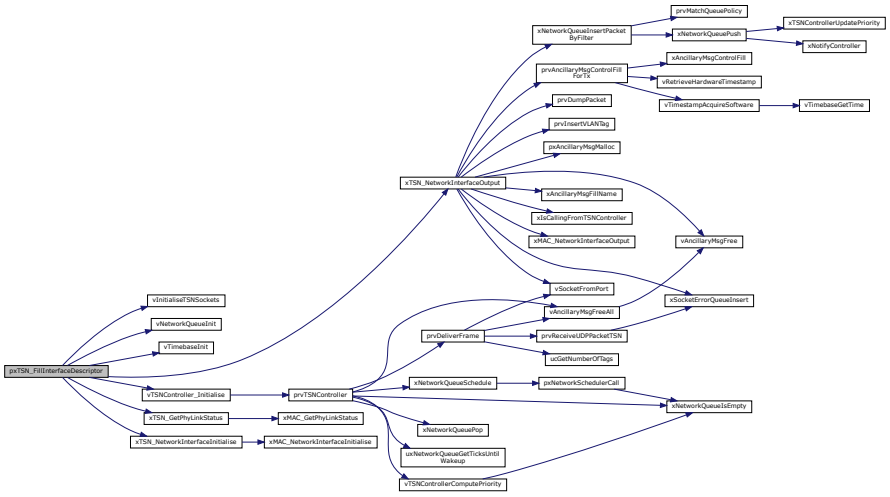
Here is the caller graph for this function:



5.40.3.7 pxTSN_FillInterfaceDescriptor()

```
NetworkInterface_t * pxTSN_FillInterfaceDescriptor (
            BaseType_t xEMACIndex,
            NetworkInterface_t * pxInterface,
            NetworkInterfaceConfig_t * pxInterfaceConfig )
```

Here is the call graph for this function:

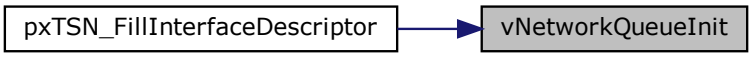


5.40.3.8 vNetworkQueueInit()

```
void vNetworkQueueInit ( )
```

Initializes the network queues and network nodes.

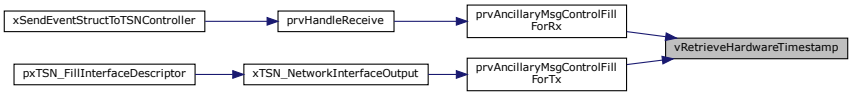
This function should be defined by the user and is project specific Here is the caller graph for this function:



5.40.3.9 vRetrieveHardwareTimestamp()

```
void vRetrieveHardwareTimestamp (
            NetworkInterface_t * pxInterface,
            NetworkBufferDescriptor_t * pxBuf,
            uint32_t * pusSec,
            uint32_t * pusNanosec )
```

Here is the caller graph for this function:



5.40.3.10 vTimebaseInit()

```
void vTimebaseInit ( )
```

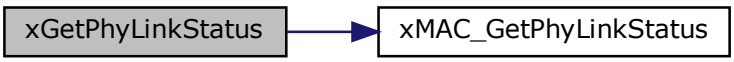
Initializes and starts the timebase.

This function should be defined by the user and is project specific. Please remember that this function is also responsible for starting the timer. Here is the caller graph for this function:

5.40.3.11 xGetPhyLinkStatus()

```
BaseType_t xGetPhyLinkStatus (
            NetworkInterface_t * pxInterface )
```

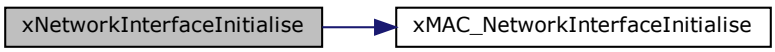
Here is the call graph for this function:



5.40.3.12 xNetworkInterfaceInitialise()

```
BaseType_t xNetworkInterfaceInitialise (
            NetworkInterface_t * pxInterface )
```

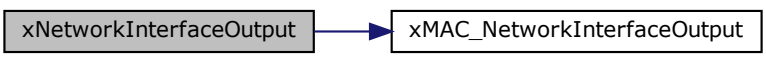
Here is the call graph for this function:



5.40.3.13 xNetworkInterfaceOutput()

```
BaseType_t xNetworkInterfaceOutput (
            NetworkInterface_t * pxInterface,
            NetworkBufferDescriptor_t *const pxBuffer,
            BaseType_t bReleaseAfterSend )
```

Here is the call graph for this function:



5.40.3.14 xSendEventStructToTSNController()

```
BaseType_t xSendEventStructToTSNController (
            const IPStackEvent_t * pxEvent,
            TickType_t uxTimeout )
```

Function to send an event structure to the TSN Controller.

This is the counterpart of the xSendEventStructToIPTask that sends the event struct to the TSN controller in place of the IP task. This is called by the network interface for handling received packets. This function is also responsible for generating the ancillary message with the packet and acquiring the timestamp if timestamping is enabled.

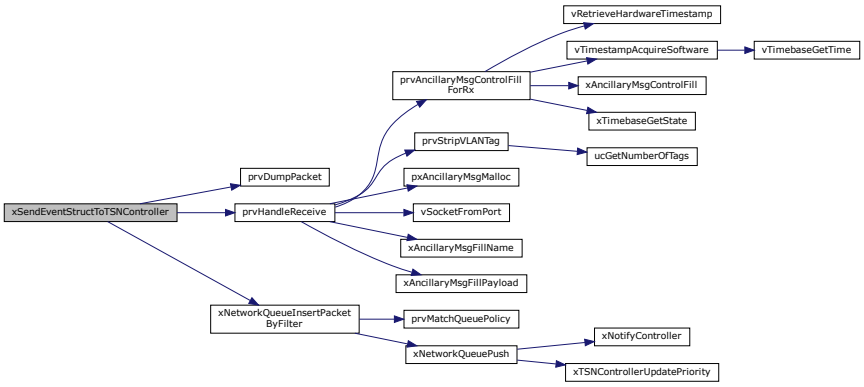
Parameters

in	*pxEvent*	Pointer to the IP stack event structure
in	*uxTimeout*	Timeout value for sending the event

Returns

pdTRUE if the event is sent successfully, pdFALSE otherwise

Here is the call graph for this function:

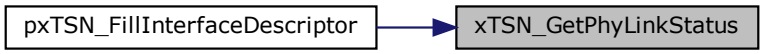


5.40.3.15 xTSN_GetPhyLinkStatus()

```
BaseType_t xTSN_GetPhyLinkStatus (
            NetworkInterface_t * pxInterface )
```

Here is the call graph for this function:

Here is the caller graph for this function:



5.40.3.16 xTSN_NetworkInterfaceInitialise()

```
BaseType_t xTSN_NetworkInterfaceInitialise (
            NetworkInterface_t * pxInterface )
```

Here is the call graph for this function:

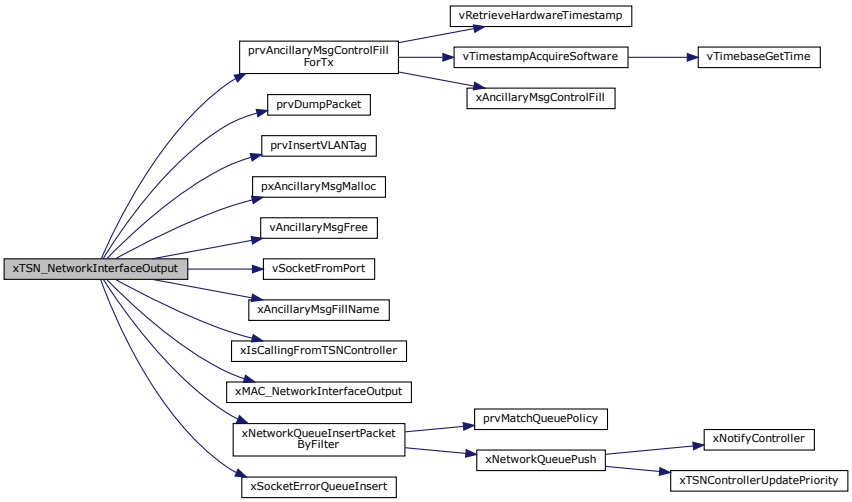
Here is the caller graph for this function:

5.40.3.17 xTSN_NetworkInterfaceOutput()

```
BaseType_t xTSN_NetworkInterfaceOutput (
            NetworkInterface_t * pxInterface,
            NetworkBufferDescriptor_t *const pxBuffer,
            BaseType_t bReleaseAfterSend )
```

The function used to send a packet.

If called from the TSN controller, forwards the packet to the original network interface, otherwise the packets is queued inside the network scheduler and the controller will take care of it in due time. Here is the call graph for this function:



Here is the caller graph for this function:

5.40.4 Variable Documentation

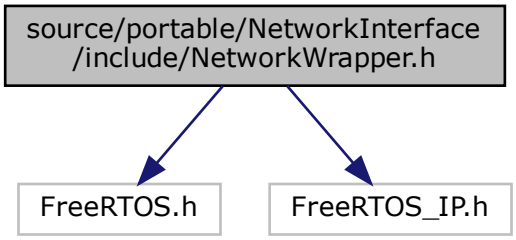
5.40.4.1 xNetworkWrapperInitialised

```
BaseType_t xNetworkWrapperInitialised = pdFALSE
```

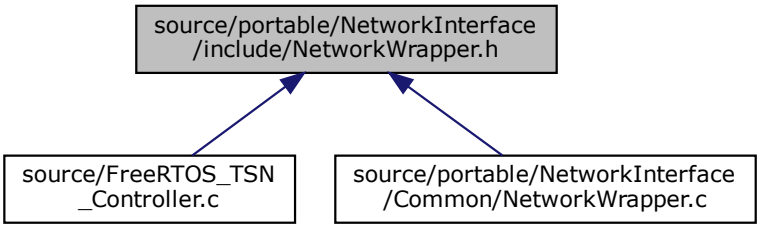
5.41 source/portable/NetworkInterface/include/NetworkWrapper.h File Reference

```
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"
```

Include dependency graph for NetworkWrapper.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xNETWORK_INTERFACE_CONFIG

Typedefs

- typedef struct xNETWORK_INTERFACE_CONFIG NetworkInterfaceConfig_t

Functions

- BaseType_t xTSN_NetworkInterfaceInitialise (NetworkInterface_t *pxInterface)
- BaseType_t xTSN_NetworkInterfaceOutput (NetworkInterface_t *pxInterface, NetworkBufferDescriptor_t *const pxBuffer, BaseType_t bReleaseAfterSend)

  The function used to send a packet.
- NetworkInterface_t * pxTSN_FillInterfaceDescriptor (BaseType_t xEMACIndex, NetworkInterface_t *pxInterface, NetworkInterfaceConfig_t *pxInterfaceConfig)
- BaseType_t xTSN_GetPhyLinkStatus (NetworkInterface_t *pxInterface)
- BaseType_t xMAC_NetworkInterfaceInitialise (NetworkInterface_t *pxInterface)

- BaseType_t xMAC_NetworkInterfaceOutput (NetworkInterface_t *pxInterface, NetworkBufferDescriptor_t *const pxBuffer, BaseType_t bReleaseAfterSend)
- BaseType_t xMAC_GetPhyLinkStatus (NetworkInterface_t *pxInterface)
- void vRetrieveHardwareTimestamp (NetworkInterface_t *pxInterface, NetworkBufferDescriptor_t *pxBuf, uint32_t *pusSec, uint32_t *pusNanosec)
- BaseType_t xSendEventStructToTSNController (const IPStackEvent_t *pxEvent, TickType_t uxTimeout)

  Function to send an event structure to the TSN Controller.

5.41.1 Typedef Documentation

5.41.1.1 NetworkInterfaceConfig_t

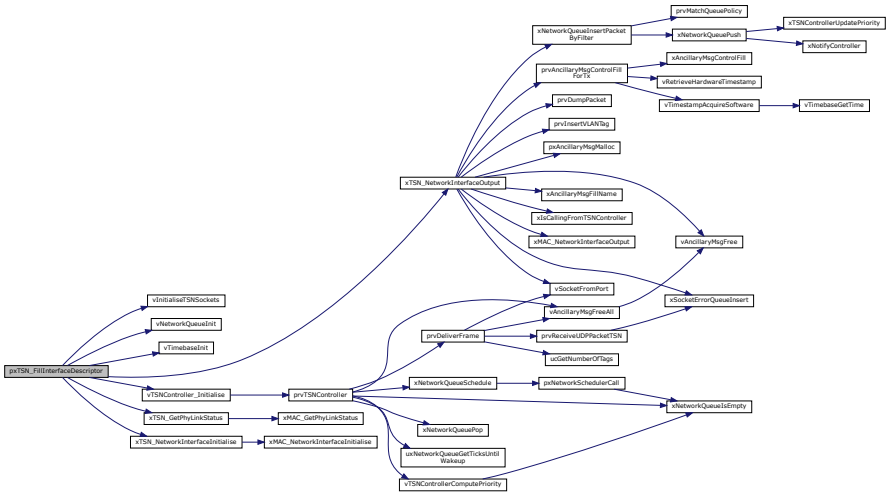
```
typedef struct xNETWORK_INTERFACE_CONFIG NetworkInterfaceConfig_t
```

5.41.2 Function Documentation

5.41.2.1 pxTSN_FillInterfaceDescriptor()

```
NetworkInterface_t * pxTSN_FillInterfaceDescriptor (
            BaseType_t xEMACIndex,
            NetworkInterface_t * pxInterface,
            NetworkInterfaceConfig_t * pxInterfaceConfig )
```

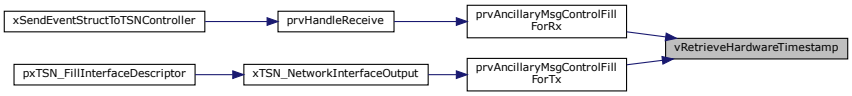
Here is the call graph for this function:



5.41.2.2 vRetrieveHardwareTimestamp()

```
void vRetrieveHardwareTimestamp (
            NetworkInterface_t * pxInterface,
            NetworkBufferDescriptor_t * pxBuf,
            uint32_t * pusSec,
            uint32_t * pusNanosec )
```

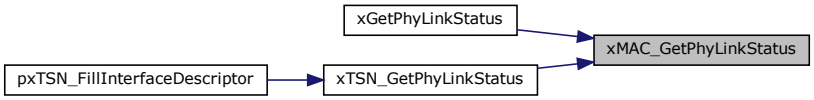
Here is the caller graph for this function:



5.41.2.3 xMAC_GetPhyLinkStatus()

```
BaseType_t xMAC_GetPhyLinkStatus (
            NetworkInterface_t * pxInterface )
```

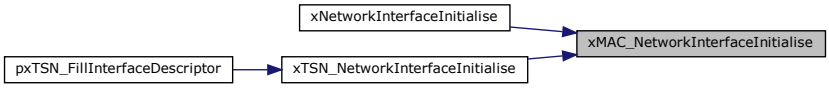
Here is the caller graph for this function:



5.41.2.4 xMAC_NetworkInterfaceInitialise()

```
BaseType_t xMAC_NetworkInterfaceInitialise (
            NetworkInterface_t * pxInterface )
```

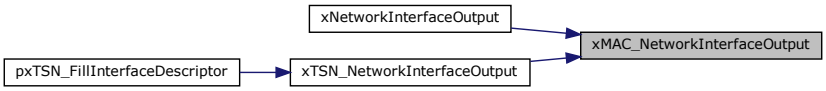
Here is the caller graph for this function:



5.41.2.5 xMAC_NetworkInterfaceOutput()

```
BaseType_t xMAC_NetworkInterfaceOutput (
            NetworkInterface_t * pxInterface,
            NetworkBufferDescriptor_t *const pxBuffer,
            BaseType_t bReleaseAfterSend )
```

Here is the caller graph for this function:



5.41.2.6 xSendEventStructToTSNController()

```
BaseType_t xSendEventStructToTSNController (
            const IPStackEvent_t * pxEvent,
            TickType_t uxTimeout )
```

Function to send an event structure to the TSN Controller.

This is the counterpart of the xSendEventStructToIPTask that sends the event struct to the TSN controller in place of the IP task. This is called by the network interface for handling received packets. This function is also responsible for generating the ancillary message with the packet and acquiring the timestamp if timestamping is enabled.

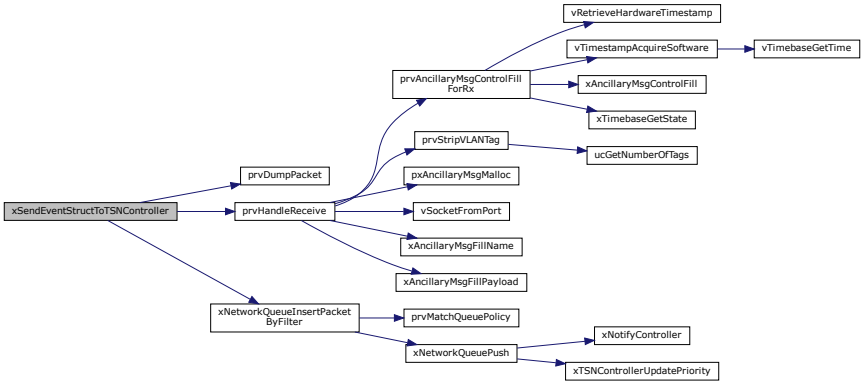
Parameters

in	*pxEvent*	Pointer to the IP stack event structure
in	*uxTimeout*	Timeout value for sending the event

Returns

pdTRUE if the event is sent successfully, pdFALSE otherwise

Here is the call graph for this function:

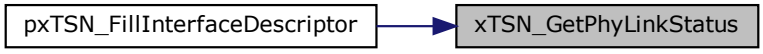


5.41.2.7 xTSN_GetPhyLinkStatus()

```
BaseType_t xTSN_GetPhyLinkStatus (
            NetworkInterface_t * pxInterface )
```

Here is the call graph for this function:

Here is the caller graph for this function:



5.41.2.8 xTSN_NetworkInterfaceInitialise()

```
BaseType_t xTSN_NetworkInterfaceInitialise (
            NetworkInterface_t * pxInterface )
```

Here is the call graph for this function:

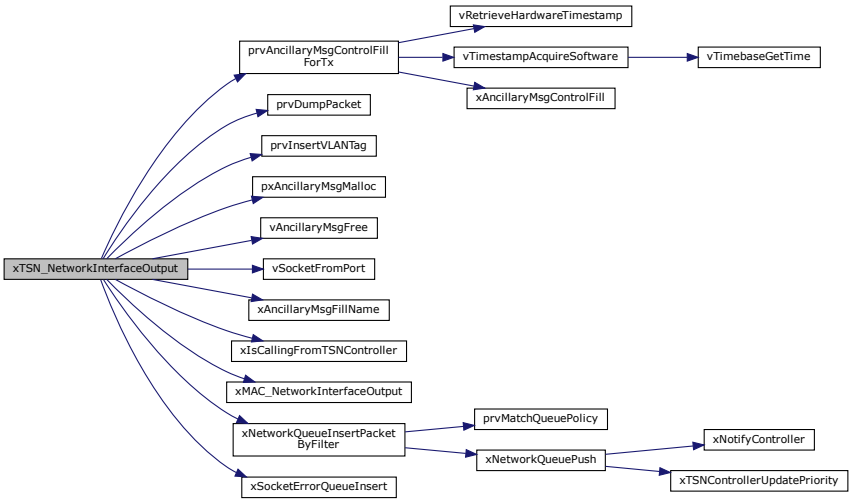
Here is the caller graph for this function:

5.41.2.9 xTSN_NetworkInterfaceOutput()

```
BaseType_t xTSN_NetworkInterfaceOutput (
            NetworkInterface_t * pxInterface,
            NetworkBufferDescriptor_t *const pxBuffer,
            BaseType_t bReleaseAfterSend )
```

The function used to send a packet.

If called from the TSN controller, forwards the packet to the original network interface, otherwise the packets is queued inside the network scheduler and the controller will take care of it in due time. Here is the call graph for this function:



Here is the caller graph for this function:

5.42 NetworkWrapper.h

Go to the documentation of this file.

```
1 #ifndef NETWORK_WRAPPER_H
2 #define NETWORK_WRAPPER_H
3
4 #include "FreeRTOS.h"
5
6 #include "FreeRTOS_IP.h"
7
8 struct xNETWORK_INTERFACE_CONFIG
9 {
10     BaseType_t xEMACIndex;
11     BaseType_t xNumTags;
12     uint16_t usVLANTag;
13     uint16_t usServiceVLANTag;
14 };
15
16 typedef struct xNETWORK_INTERFACE_CONFIG NetworkInterfaceConfig_t;
17
18 /* Definitions used to create the API exported to the upper layers
19 */
20 BaseType_t xTSN_NetworkInterfaceInitialise( NetworkInterface_t * pxInterface );
21
22 BaseType_t xTSN_NetworkInterfaceOutput( NetworkInterface_t * pxInterface,
23                                         NetworkBufferDescriptor_t * const pxBuffer,
24                                         BaseType_t bReleaseAfterSend );
25
26 NetworkInterface_t * pxTSN_FillInterfaceDescriptor( BaseType_t xEMACIndex,
27                                                     NetworkInterface_t * pxInterface,
28                                                     NetworkInterfaceConfig_t * pxInterfaceConfig );
29
30 BaseType_t xTSN_GetPhyLinkStatus( NetworkInterface_t * pxInterface );
31
32
33 /* Definitions used at this level to interface
34 * the lower MAC layer
35 */
36 BaseType_t xMAC_NetworkInterfaceInitialise( NetworkInterface_t * pxInterface );
37
38 BaseType_t xMAC_NetworkInterfaceOutput( NetworkInterface_t * pxInterface,
39                                         NetworkBufferDescriptor_t * const pxBuffer,
40                                         BaseType_t bReleaseAfterSend );
41
42 #if defined( ipconfigIPv4_BACKWARD_COMPATIBLE ) && ( ipconfigIPv4_BACKWARD_COMPATIBLE == 1 )
43     NetworkInterface_t * pxMAC_FillInterfaceDescriptor( BaseType_t xEMACIndex,
44                                                         NetworkInterface_t * pxInterface );
45 #endif
46
47 BaseType_t xMAC_GetPhyLinkStatus( NetworkInterface_t * pxInterface );
48
49
50 void vRetrieveHardwareTimestamp( NetworkInterface_t * pxInterface,
51                                  NetworkBufferDescriptor_t * pxBuf,
52                                  uint32_t * pusSec,
53                                  uint32_t * pusNanosec );
54
55 BaseType_t xSendEventStructToTSNController( const IPStackEvent_t * pxEvent,
56                                             TickType_t uxTimeout );
57
58 #endif /* NETWORK_WRAPPER_H */
```

Index