

FreRTOS TSN Compatibility Layer

Generated by Doxygen 1.9.4

| | |
|--|----------|
| 1 FreeRTOS TSN Compatibility Layer | 1 |
| 1.1 Setting up the project | 1 |
| 1.2 Configuration | 1 |
| 1.2.1 Network scheduler | 2 |
| 1.2.2 Timebase | 2 |
| 1.2.3 TSN Config | 2 |
| 1.3 TSN Sockets | 2 |
| 2 Data Structure Index | 3 |
| 2.1 Data Structures | 3 |
| 3 File Index | 5 |
| 3.1 File List | 5 |
| 4 Data Structure Documentation | 7 |
| 4.1 cmsghdr Struct Reference | 7 |
| 4.1.1 Field Documentation | 7 |
| 4.1.1.1 cmsg_len | 7 |
| 4.1.1.2 cmsg_level | 7 |
| 4.1.1.3 cmsg_type | 7 |
| 4.2 freertos_scm_timestamping Struct Reference | 8 |
| 4.2.1 Field Documentation | 8 |
| 4.2.1.1 ts | 8 |
| 4.3 freertos_timespec Struct Reference | 8 |
| 4.3.1 Detailed Description | 9 |
| 4.3.2 Field Documentation | 9 |
| 4.3.2.1 tv_nsec | 9 |
| 4.3.2.2 tv_sec | 9 |
| 4.4 iovector Struct Reference | 9 |
| 4.4.1 Field Documentation | 9 |
| 4.4.1.1 iov_base | 9 |
| 4.4.1.2 iov_len | 10 |
| 4.5 msghdr Struct Reference | 10 |
| 4.5.1 Field Documentation | 10 |
| 4.5.1.1 msg_control | 10 |
| 4.5.1.2 msg_controllen | 11 |
| 4.5.1.3 msg_flags | 11 |
| 4.5.1.4 msg_iov | 11 |
| 4.5.1.5 msg_iovlen | 11 |
| 4.5.1.6 msg_name | 11 |
| 4.5.1.7 msg_namelen | 11 |
| 4.6 sock_extended_err Struct Reference | 11 |
| 4.6.1 Field Documentation | 12 |

| | |
|---|----|
| 4.6.1.1 ee_code | 12 |
| 4.6.1.2 ee_data | 12 |
| 4.6.1.3 ee_errno | 12 |
| 4.6.1.4 ee_info | 12 |
| 4.6.1.5 ee_origin | 12 |
| 4.6.1.6 ee_pad | 13 |
| 4.6.1.7 ee_type | 13 |
| 4.7 struct Struct Reference | 13 |
| 4.7.1 Field Documentation | 13 |
| 4.7.1.1 usFrameType | 13 |
| 4.7.1.2 xDestinationAddress | 14 |
| 4.7.1.3 xSourceAddress | 14 |
| 4.7.1.4 xVLANCTag | 14 |
| 4.7.1.5 xVLANSTag | 14 |
| 4.8 xNETQUEUE Struct Reference | 14 |
| 4.8.1 Detailed Description | 15 |
| 4.8.2 Field Documentation | 15 |
| 4.8.2.1 cName | 15 |
| 4.8.2.2 ePolicy | 15 |
| 4.8.2.3 fnFilter | 15 |
| 4.8.2.4 uxIPV | 15 |
| 4.8.2.5 xQueue | 16 |
| 4.9 xNETQUEUE_ITEM Struct Reference | 16 |
| 4.9.1 Detailed Description | 16 |
| 4.9.2 Field Documentation | 17 |
| 4.9.2.1 eEventType | 17 |
| 4.9.2.2 pxBuf | 17 |
| 4.9.2.3 pxMsg | 17 |
| 4.9.2.4 xReleaseAfterSend | 17 |
| 4.10 xNETQUEUE_NODE Struct Reference | 17 |
| 4.10.1 Detailed Description | 18 |
| 4.10.2 Field Documentation | 18 |
| 4.10.2.1 pvScheduler | 18 |
| 4.10.2.2 pxNext | 18 |
| 4.10.2.3 pxQueue | 18 |
| 4.10.2.4 ucNumChildren | 18 |
| 4.11 xNETWORK_INTERFACE_CONFIG Struct Reference | 19 |
| 4.11.1 Field Documentation | 19 |
| 4.11.1.1 usServiceVLANTag | 19 |
| 4.11.1.2 usVLANTag | 19 |
| 4.11.1.3 xEMACIndex | 19 |
| 4.11.1.4 xNumTags | 19 |

| | |
|--|----|
| 4.12 xQUEUE_LIST Struct Reference | 20 |
| 4.12.1 Detailed Description | 20 |
| 4.12.2 Field Documentation | 20 |
| 4.12.2.1 pxNext | 20 |
| 4.12.2.2 pxQueue | 21 |
| 4.13 xSCHEDULER_CBS Struct Reference | 21 |
| 4.13.1 Field Documentation | 21 |
| 4.13.1.1 uxBandwidth | 21 |
| 4.13.1.2 uxMaxCredit | 21 |
| 4.13.1.3 uxNextActivation | 22 |
| 4.13.1.4 xScheduler | 22 |
| 4.14 xSCHEDULER_FIFO Struct Reference | 22 |
| 4.14.1 Field Documentation | 22 |
| 4.14.1.1 xScheduler | 22 |
| 4.15 xSCHEDULER_GENERIC Struct Reference | 23 |
| 4.15.1 Detailed Description | 23 |
| 4.15.2 Field Documentation | 23 |
| 4.15.2.1 fnReady | 24 |
| 4.15.2.2 fnSelect | 24 |
| 4.15.2.3 pxOwner | 24 |
| 4.15.2.4 ucAttributes | 24 |
| 4.15.2.5 usSize | 24 |
| 4.16 xSCHEDULER_PRIO Struct Reference | 24 |
| 4.16.1 Field Documentation | 25 |
| 4.16.1.1 xScheduler | 25 |
| 4.17 xSCHEDULER_RR Struct Reference | 25 |
| 4.17.1 Field Documentation | 25 |
| 4.17.1.1 xScheduler | 26 |
| 4.18 xTIMEBASE Struct Reference | 26 |
| 4.18.1 Field Documentation | 26 |
| 4.18.1.1 fnAdjTime | 26 |
| 4.18.1.2 fnGetTime | 27 |
| 4.18.1.3 fnSetTime | 27 |
| 4.18.1.4 fnStart | 27 |
| 4.18.1.5 fnStop | 27 |
| 4.19 xTSN_SOCKET Struct Reference | 27 |
| 4.19.1 Field Documentation | 27 |
| 4.19.1.1 ucDSCClass | 28 |
| 4.19.1.2 ulTSFlags | 28 |
| 4.19.1.3 xBaseSocket | 28 |
| 4.19.1.4 xBoundSocketListItem | 28 |
| 4.19.1.5 xErrQueue | 28 |

| | |
|---|-----------|
| 4.19.1.6 xRecvTask | 28 |
| 4.19.1.7 xSendTask | 28 |
| 4.20 xVLAN_TAG Struct Reference | 29 |
| 4.20.1 Field Documentation | 29 |
| 4.20.1.1 usTCI | 29 |
| 4.20.1.2 usTPID | 29 |
| 5 File Documentation | 31 |
| 5.1 README.md File Reference | 31 |
| 5.2 source/FreeRTOS_TSN_Ancillary.c File Reference | 31 |
| 5.2.1 Detailed Description | 32 |
| 5.2.2 Function Documentation | 32 |
| 5.2.2.1 __CMSG_NXTHDR() | 32 |
| 5.2.2.2 pxAncillaryMsgMalloc() | 32 |
| 5.2.2.3 vAncillaryMsgFree() | 33 |
| 5.2.2.4 vAncillaryMsgFreeAll() | 33 |
| 5.2.2.5 vAncillaryMsgFreeControl() | 34 |
| 5.2.2.6 vAncillaryMsgFreeName() | 34 |
| 5.2.2.7 vAncillaryMsgFreePayload() | 35 |
| 5.2.2.8 xAncillaryMsgControlFill() | 35 |
| 5.2.2.9 xAncillaryMsgControlFillSingle() | 36 |
| 5.2.2.10 xAncillaryMsgFillName() | 36 |
| 5.2.2.11 xAncillaryMsgFillPayload() | 37 |
| 5.3 source/FreeRTOS_TSN_Controller.c File Reference | 38 |
| 5.3.1 Detailed Description | 39 |
| 5.3.2 Macro Definition Documentation | 39 |
| 5.3.2.1 controllerTSN_TASK_BASE_PRIO | 39 |
| 5.3.3 Function Documentation | 39 |
| 5.3.3.1 prvDeliverFrame() | 39 |
| 5.3.3.2 prvReceiveUDPPacketTSN() | 40 |
| 5.3.3.3 prvTSNController() | 41 |
| 5.3.3.4 vTSNController_Initialise() | 42 |
| 5.3.3.5 vTSNControllerComputePriority() | 43 |
| 5.3.3.6 xIsCallingFromTSNController() | 43 |
| 5.3.3.7 xNotifyController() | 44 |
| 5.3.3.8 xTSNControllerUpdatePriority() | 44 |
| 5.3.4 Variable Documentation | 45 |
| 5.3.4.1 pxNetworkQueueList | 45 |
| 5.3.4.2 xTSNControllerHandle | 45 |
| 5.4 source/FreeRTOS_TSN_DS.c File Reference | 45 |
| 5.4.1 Detailed Description | 46 |
| 5.4.2 Function Documentation | 46 |

| | |
|--|----|
| 5.4.2.1 prvGetIPVersionAndOffset() | 46 |
| 5.4.2.2 ucDSCClassGet() | 46 |
| 5.4.2.3 xDSCClassSet() | 47 |
| 5.5 source/FreeRTOS_TSN_NetworkScheduler.c File Reference | 48 |
| 5.5.1 Detailed Description | 49 |
| 5.5.2 Function Documentation | 49 |
| 5.5.2.1 prvMatchQueuePolicy() | 49 |
| 5.5.2.2 vNetworkQueueListAdd() | 49 |
| 5.5.2.3 xNetworkQueueAssignRoot() | 50 |
| 5.5.2.4 xNetworkQueueInsertPacketByFilter() | 50 |
| 5.5.2.5 xNetworkQueueInsertPacketByName() | 51 |
| 5.5.2.6 xNetworkQueuePop() | 52 |
| 5.5.2.7 xNetworkQueuePush() | 53 |
| 5.5.2.8 xNetworkQueueSchedule() | 54 |
| 5.5.3 Variable Documentation | 54 |
| 5.5.3.1 pxNetworkQueueList | 54 |
| 5.5.3.2 pxNetworkQueueRoot | 54 |
| 5.5.3.3 uxNumQueues | 55 |
| 5.6 source/FreeRTOS_TSN_NetworkSchedulerBlock.c File Reference | 55 |
| 5.6.1 Detailed Description | 56 |
| 5.6.2 Function Documentation | 56 |
| 5.6.2.1 prvAlwaysReady() | 56 |
| 5.6.2.2 prvSelectFirst() | 56 |
| 5.6.2.3 pxNetworkSchedulerCall() | 57 |
| 5.6.2.4 pxPeekNextPacket() | 58 |
| 5.6.2.5 uxNetworkQueueGetTicksUntilWakeup() | 59 |
| 5.6.2.6 vNetworkQueueAddWakeupEvent() | 59 |
| 5.6.2.7 xNetworkSchedulerLinkChild() | 60 |
| 5.6.2.8 xNetworkSchedulerLinkQueue() | 60 |
| 5.6.3 Variable Documentation | 61 |
| 5.6.3.1 uxNextWakeup | 61 |
| 5.7 source/FreeRTOS_TSN_NetworkSchedulerQueue.c File Reference | 61 |
| 5.7.1 Detailed Description | 62 |
| 5.7.2 Function Documentation | 62 |
| 5.7.2.1 prvAlwaysTrue() | 62 |
| 5.7.2.2 prvDefaultPacketHandler() | 62 |
| 5.7.2.3 uxNetworkQueuePacketsWaiting() | 63 |
| 5.7.2.4 xNetworkQueueIsEmpty() | 63 |
| 5.8 source/FreeRTOS_TSN_Sockets.c File Reference | 64 |
| 5.8.1 Detailed Description | 65 |
| 5.8.2 Macro Definition Documentation | 65 |
| 5.8.2.1 tsnsocketGET_SOCKET_PORT | 66 |

| | |
|---|----|
| 5.8.2.2 tsnsocketSET_SOCKET_PORT | 66 |
| 5.8.2.3 tsnsocketSOCKET_IS_BOUND | 66 |
| 5.8.3 Function Documentation | 66 |
| 5.8.3.1 FreeRTOS_TSN_bind() | 66 |
| 5.8.3.2 FreeRTOS_TSN_closesocket() | 67 |
| 5.8.3.3 FreeRTOS_TSN_recvfrom() | 67 |
| 5.8.3.4 FreeRTOS_TSN_recvmsg() | 68 |
| 5.8.3.5 FreeRTOS_TSN_sendto() | 69 |
| 5.8.3.6 FreeRTOS_TSN_setsockopt() | 70 |
| 5.8.3.7 FreeRTOS_TSN_socket() | 70 |
| 5.8.3.8 prvMoveToStartOfPayload() | 71 |
| 5.8.3.9 prvPrepareBufferUDIPv4() | 71 |
| 5.8.3.10 prvPrepareBufferUDIPv6() | 72 |
| 5.8.3.11 vInitialiseTSNSockets() | 73 |
| 5.8.3.12 vSocketFromPort() | 73 |
| 5.8.3.13 xSocketErrorQueueInsert() | 74 |
| 5.8.4 Variable Documentation | 74 |
| 5.8.4.1 xTSNBoundUDPSocketList | 74 |
| 5.9 source/FreeRTOS_TSN_Timebase.c File Reference | 74 |
| 5.9.1 Detailed Description | 75 |
| 5.9.2 Macro Definition Documentation | 75 |
| 5.9.2.1 NS_IN_ONE_SEC | 75 |
| 5.9.3 Function Documentation | 76 |
| 5.9.3.1 vTimebaseAdjTime() | 76 |
| 5.9.3.2 vTimebaseGetTime() | 76 |
| 5.9.3.3 vTimebaseSetTime() | 76 |
| 5.9.3.4 vTimebaseStart() | 77 |
| 5.9.3.5 xTimebaseGetState() | 77 |
| 5.9.3.6 xTimebaseHandleSet() | 77 |
| 5.9.3.7 xTimespecCmp() | 78 |
| 5.9.3.8 xTimespecDiff() | 78 |
| 5.9.3.9 xTimespecDiv() | 78 |
| 5.9.3.10 xTimespecSum() | 79 |
| 5.9.4 Variable Documentation | 80 |
| 5.9.4.1 xTimebaseHandle | 80 |
| 5.9.4.2 xTimebaseState | 80 |
| 5.10 source/FreeRTOS_TSN_Timestamp.c File Reference | 80 |
| 5.10.1 Detailed Description | 81 |
| 5.10.2 Function Documentation | 81 |
| 5.10.2.1 vTimestampAcquireSoftware() | 81 |
| 5.11 source/FreeRTOS_TSN_VLANTags.c File Reference | 82 |
| 5.11.1 Detailed Description | 83 |

| | |
|---|-----|
| 5.11.2 Function Documentation | 83 |
| 5.11.2.1 prvGetVLANCTag() | 84 |
| 5.11.2.2 prvGetVLANSTag() | 84 |
| 5.11.2.3 prvPrepareAndGetVLANCTag() | 85 |
| 5.11.2.4 prvPrepareAndGetVLANSTag() | 86 |
| 5.11.2.5 ucGetNumberOfTags() | 87 |
| 5.11.2.6 xVLANCTagCheckClass() | 88 |
| 5.11.2.7 xVLANCTagGetDEI() | 89 |
| 5.11.2.8 xVLANCTagGetPCP() | 90 |
| 5.11.2.9 xVLANCTagGetVID() | 90 |
| 5.11.2.10 xVLANCTagSetDEI() | 91 |
| 5.11.2.11 xVLANCTagSetPCP() | 92 |
| 5.11.2.12 xVLANCTagSetVID() | 93 |
| 5.11.2.13 xVLANSTagCheckClass() | 93 |
| 5.11.2.14 xVLANSTagGetDEI() | 94 |
| 5.11.2.15 xVLANSTagGetPCP() | 95 |
| 5.11.2.16 xVLANSTagGetVID() | 95 |
| 5.11.2.17 xVLANSTagSetDEI() | 96 |
| 5.11.2.18 xVLANSTagSetPCP() | 97 |
| 5.11.2.19 xVLANSTagSetVID() | 98 |
| 5.12 source/include/FreeRTOS_TSN_Ancillary.h File Reference | 98 |
| 5.12.1 Macro Definition Documentation | 100 |
| 5.12.1.1 __CMSG_FIRSTHDR | 100 |
| 5.12.1.2 CMSG_ALIGN | 100 |
| 5.12.1.3 CMSG_DATA | 100 |
| 5.12.1.4 CMSG_FIRSTHDR | 100 |
| 5.12.1.5 CMSG_LEN | 101 |
| 5.12.1.6 CMSG_NXTHDR | 101 |
| 5.12.1.7 CMSG_SPACE | 101 |
| 5.12.1.8 pdFREERTOS_ERRNO_ENOMSG | 101 |
| 5.12.1.9 SO_EE_ORIGIN_ICMP | 101 |
| 5.12.1.10 SO_EE_ORIGIN_ICMP6 | 101 |
| 5.12.1.11 SO_EE_ORIGIN_LOCAL | 101 |
| 5.12.1.12 SO_EE_ORIGIN_NONE | 102 |
| 5.12.1.13 SO_EE_ORIGIN_TIMESTAMPING | 102 |
| 5.12.1.14 SO_EE_ORIGIN_TXSTATUS | 102 |
| 5.12.1.15 SO_EE_ORIGIN_TXTIME | 102 |
| 5.12.1.16 SO_EE_ORIGIN_ZEROCOPY | 102 |
| 5.12.2 Function Documentation | 102 |
| 5.12.2.1 __CMSG_NXTHDR() | 102 |
| 5.12.2.2 pxAncillaryMsgMalloc() | 103 |
| 5.12.2.3 vAncillaryMsgFree() | 103 |

| | |
|--|-----|
| 5.12.2.4 vAncillaryMsgFreeAll() | 104 |
| 5.12.2.5 vAncillaryMsgFreeControl() | 104 |
| 5.12.2.6 vAncillaryMsgFreeName() | 105 |
| 5.12.2.7 vAncillaryMsgFreePayload() | 105 |
| 5.12.2.8 xAncillaryMsgControlFill() | 105 |
| 5.12.2.9 xAncillaryMsgControlFillSingle() | 106 |
| 5.12.2.10 xAncillaryMsgFillName() | 106 |
| 5.12.2.11 xAncillaryMsgFillPayload() | 107 |
| 5.13 FreeRTOS_TSN_Ancillary.h | 107 |
| 5.14 source/include/FreeRTOS_TSN_Controller.h File Reference | 109 |
| 5.14.1 Function Documentation | 110 |
| 5.14.1.1 vTSNController_Initialise() | 110 |
| 5.14.1.2 vTSNControllerComputePriority() | 111 |
| 5.14.1.3 xIsCallingFromTSNController() | 111 |
| 5.14.1.4 xNotifyController() | 112 |
| 5.14.1.5 xTSNControllerUpdatePriority() | 112 |
| 5.15 FreeRTOS_TSN_Controller.h | 113 |
| 5.16 source/include/FreeRTOS_TSN_DS.h File Reference | 113 |
| 5.16.1 Macro Definition Documentation | 114 |
| 5.16.1.1 diffservCLASS_AFxy | 114 |
| 5.16.1.2 diffservCLASS_CSx | 114 |
| 5.16.1.3 diffservCLASS_DF | 114 |
| 5.16.1.4 diffservCLASS_DSCP_CUSTOM | 115 |
| 5.16.1.5 diffservCLASS_EF | 115 |
| 5.16.1.6 diffservCLASS_LE | 115 |
| 5.16.1.7 diffservGET_DSCLASS_IPv4 | 115 |
| 5.16.1.8 diffservGET_DSCLASS_IPv6 | 115 |
| 5.16.1.9 diffservSET_DSCLASS_IPv4 | 115 |
| 5.16.1.10 diffservSET_DSCLASS_IPv6 | 116 |
| 5.16.2 Function Documentation | 116 |
| 5.16.2.1 ucDSCClassGet() | 116 |
| 5.16.2.2 xDSCClassSet() | 117 |
| 5.17 FreeRTOS_TSN_DS.h | 117 |
| 5.18 source/include/FreeRTOS_TSN_NetworkScheduler.h File Reference | 118 |
| 5.18.1 Typedef Documentation | 119 |
| 5.18.1.1 NetworkQueueList_t | 119 |
| 5.18.2 Function Documentation | 119 |
| 5.18.2.1 pxNetworkQueueFindByName() | 119 |
| 5.18.2.2 vNetworkQueueInit() | 120 |
| 5.18.2.3 vNetworkQueueListAdd() | 120 |
| 5.18.2.4 xNetworkQueueAssignRoot() | 120 |
| 5.18.2.5 xNetworkQueueInsertPacketByFilter() | 120 |

| | |
|---|-----|
| 5.18.2.6 xNetworkQueueInsertPacketByName() | 121 |
| 5.18.2.7 xNetworkQueuePop() | 122 |
| 5.18.2.8 xNetworkQueuePush() | 123 |
| 5.18.2.9 xNetworkQueueSchedule() | 124 |
| 5.19 FreeRTOS_TSN_NetworkScheduler.h | 124 |
| 5.20 source/include/FreeRTOS_TSN_NetworkSchedulerBlock.h File Reference | 125 |
| 5.20.1 Macro Definition Documentation | 126 |
| 5.20.1.1 netschedCALL_READY_FROM_NODE | 126 |
| 5.20.1.2 netschedCALL_SELECT_FROM_NODE | 126 |
| 5.20.2 Typedef Documentation | 127 |
| 5.20.2.1 NetworkNode_t | 127 |
| 5.20.2.2 ReadyQueueFunction_t | 127 |
| 5.20.2.3 SelectQueueFunction_t | 127 |
| 5.20.3 Function Documentation | 127 |
| 5.20.3.1 pxNetworkSchedulerCall() | 127 |
| 5.20.3.2 pxPeekNextPacket() | 128 |
| 5.20.3.3 uxNetworkQueueGetTicksUntilWakeup() | 129 |
| 5.20.3.4 vNetworkQueueAddWakeupEvent() | 129 |
| 5.20.3.5 xNetworkSchedulerLinkChild() | 130 |
| 5.20.3.6 xNetworkSchedulerLinkQueue() | 130 |
| 5.21 FreeRTOS_TSN_NetworkSchedulerBlock.h | 130 |
| 5.22 source/include/FreeRTOS_TSN_NetworkSchedulerQueue.h File Reference | 131 |
| 5.22.1 Typedef Documentation | 133 |
| 5.22.1.1 FilterFunction_t | 133 |
| 5.22.1.2 NetworkQueue_t | 133 |
| 5.22.1.3 NetworkQueueItem_t | 133 |
| 5.22.1.4 PacketHandleFunction_t | 133 |
| 5.22.2 Enumeration Type Documentation | 133 |
| 5.22.2.1 eQueuePolicy_t | 133 |
| 5.22.3 Function Documentation | 134 |
| 5.22.3.1 uxNetworkQueuePacketsWaiting() | 134 |
| 5.22.3.2 xNetworkQueueIsEmpty() | 134 |
| 5.23 FreeRTOS_TSN_NetworkSchedulerQueue.h | 135 |
| 5.24 source/include/FreeRTOS_TSN_Sockets.h File Reference | 136 |
| 5.24.1 Macro Definition Documentation | 137 |
| 5.24.1.1 FREERTOS_IP_RECVERR | 138 |
| 5.24.1.2 FREERTOS_IPV6_RECVERR | 138 |
| 5.24.1.3 FREERTOS_MSG_ERRQUEUE | 138 |
| 5.24.1.4 FREERTOS_SCM_TIMESTAMP | 138 |
| 5.24.1.5 FREERTOS_SCM_TIMESTAMPING | 138 |
| 5.24.1.6 FREERTOS_SCM_TIMESTAMPNS | 138 |
| 5.24.1.7 FREERTOS_SO_DS_CLASS | 138 |

| | |
|--|-----|
| 5.24.1.8 FREERTOS_SO_TIMESTAMP | 138 |
| 5.24.1.9 FREERTOS_SO_TIMESTAMP_OLD | 139 |
| 5.24.1.10 FREERTOS_SO_TIMESTAMPING | 139 |
| 5.24.1.11 FREERTOS_SO_TIMESTAMPING_OLD | 139 |
| 5.24.1.12 FREERTOS_SO_TIMESTAMPNS | 139 |
| 5.24.1.13 FREERTOS_SO_TIMESTAMPNS_OLD | 139 |
| 5.24.1.14 FREERTOS_SOL_IP | 139 |
| 5.24.1.15 FREERTOS_SOL_IPV6 | 139 |
| 5.24.1.16 FREERTOS_SOL_SOCKET | 139 |
| 5.24.1.17 FREERTOS_TSN_INVALID_SOCKET | 140 |
| 5.24.2 Typedef Documentation | 140 |
| 5.24.2.1 FreeRTOS_TSN_Socket_t | 140 |
| 5.24.2.2 TSNSocket_t | 140 |
| 5.24.3 Enumeration Type Documentation | 140 |
| 5.24.3.1 anonymous enum | 140 |
| 5.24.3.2 anonymous enum | 141 |
| 5.24.4 Function Documentation | 141 |
| 5.24.4.1 FreeRTOS_TSN_bind() | 141 |
| 5.24.4.2 FreeRTOS_TSN_closesocket() | 141 |
| 5.24.4.3 FreeRTOS_TSN_recvfrom() | 142 |
| 5.24.4.4 FreeRTOS_TSN_recvmsg() | 143 |
| 5.24.4.5 FreeRTOS_TSN_sendto() | 144 |
| 5.24.4.6 FreeRTOS_TSN_setsockopt() | 144 |
| 5.24.4.7 FreeRTOS_TSN_socket() | 145 |
| 5.24.4.8 vInitialiseTSNSockets() | 145 |
| 5.24.4.9 vSocketFromPort() | 146 |
| 5.24.4.10 xSocketErrorQueueInsert() | 146 |
| 5.25 FreeRTOS_TSN_Sockets.h | 146 |
| 5.26 source/include/FreeRTOS_TSN_Timebase.h File Reference | 148 |
| 5.26.1 Typedef Documentation | 150 |
| 5.26.1.1 TimeBaseAdjTimeFunction_t | 150 |
| 5.26.1.2 TimeBaseGetTimeFunction_t | 150 |
| 5.26.1.3 TimebaseHandle_t | 150 |
| 5.26.1.4 TimeBaseSetTimeFunction_t | 151 |
| 5.26.1.5 TimeBaseStartFunction_t | 151 |
| 5.26.1.6 TimeBaseStopFunction_t | 151 |
| 5.26.2 Enumeration Type Documentation | 151 |
| 5.26.2.1 eTimebaseState_t | 151 |
| 5.26.3 Function Documentation | 151 |
| 5.26.3.1 vTimebaseAdjTime() | 151 |
| 5.26.3.2 vTimebaseGetTime() | 152 |
| 5.26.3.3 vTimebaseInit() | 153 |

| | |
|---|-----|
| 5.26.3.4 vTimebaseSetTime() | 153 |
| 5.26.3.5 vTimebaseStart() | 153 |
| 5.26.3.6 vTimebaseStop() | 154 |
| 5.26.3.7 xTimebaseGetState() | 154 |
| 5.26.3.8 xTimebaseHandleSet() | 154 |
| 5.26.3.9 xTimespecCmp() | 154 |
| 5.26.3.10 xTimespecDiff() | 155 |
| 5.26.3.11 xTimespecDiv() | 155 |
| 5.26.3.12 xTimespecSum() | 156 |
| 5.27 FreeRTOS_TSN_Timebase.h | 156 |
| 5.28 source/include/FreeRTOS_TSN_Timestamp.h File Reference | 157 |
| 5.28.1 Function Documentation | 158 |
| 5.28.1.1 vTimestampAcquireSoftware() | 159 |
| 5.29 FreeRTOS_TSN_Timestamp.h | 160 |
| 5.30 source/include/FreeRTOS_TSN_VLANTags.h File Reference | 160 |
| 5.30.1 Macro Definition Documentation | 162 |
| 5.30.1.1 vlantagCLASS_0 | 162 |
| 5.30.1.2 vlantagCLASS_1 | 163 |
| 5.30.1.3 vlantagCLASS_2 | 163 |
| 5.30.1.4 vlantagCLASS_3 | 163 |
| 5.30.1.5 vlantagCLASS_4 | 163 |
| 5.30.1.6 vlantagCLASS_5 | 163 |
| 5.30.1.7 vlantagCLASS_6 | 163 |
| 5.30.1.8 vlantagCLASS_7 | 163 |
| 5.30.1.9 vlantagDEI_BIT_MASK | 163 |
| 5.30.1.10 vlantagETH_TAG_OFFSET | 164 |
| 5.30.1.11 vlantagGET_DEI_FROM_TCI | 164 |
| 5.30.1.12 vlantagGET_PCP_FROM_TCI | 164 |
| 5.30.1.13 vlantagGET_VID_FROM_TCI | 164 |
| 5.30.1.14 vlantagPCP_BIT_MASK | 164 |
| 5.30.1.15 vlantagSET_DEI_FROM_TCI | 164 |
| 5.30.1.16 vlantagSET_PCP_FROM_TCI | 165 |
| 5.30.1.17 vlantagSET_VID_FROM_TCI | 165 |
| 5.30.1.18 vlantagTPID_DEFAULT | 165 |
| 5.30.1.19 vlantagTPID_DOUBLE_TAG | 165 |
| 5.30.1.20 vlantagVID_BIT_MASK | 165 |
| 5.30.1.21 xVLANTagCheckClass | 165 |
| 5.30.1.22 xVLANTagGetDEI | 166 |
| 5.30.1.23 xVLANTagGetPCP | 166 |
| 5.30.1.24 xVLANTagGetVID | 166 |
| 5.30.1.25 xVLANTagSetDEI | 166 |
| 5.30.1.26 xVLANTagSetPCP | 166 |

| | |
|--|-----|
| 5.30.1.27 xVLANTagSetVID | 166 |
| 5.30.2 Typedef Documentation | 166 |
| 5.30.2.1 TaggedEthernetHeader_t | 166 |
| 5.30.3 Function Documentation | 166 |
| 5.30.3.1 ucGetNumberOfTags() | 166 |
| 5.30.3.2 xVLANTagSetDEI() | 167 |
| 5.30.3.3 xVLANTagSetPCP() | 168 |
| 5.30.3.4 xVLANTagSetVID() | 169 |
| 5.30.3.5 xVLANSTagCheckClass() | 169 |
| 5.30.3.6 xVLANSTagGetDEI() | 170 |
| 5.30.3.7 xVLANSTagGetPCP() | 171 |
| 5.30.3.8 xVLANSTagGetVID() | 171 |
| 5.30.3.9 xVLANSTagSetDEI() | 172 |
| 5.30.3.10 xVLANSTagSetPCP() | 173 |
| 5.30.3.11 xVLANSTagSetVID() | 174 |
| 5.30.4 Variable Documentation | 174 |
| 5.30.4.1 usFrameType | 174 |
| 5.30.4.2 xDestinationAddress | 174 |
| 5.30.4.3 xSourceAddress | 175 |
| 5.30.4.4 xVLANTag | 175 |
| 5.31 FreeRTOS_TSN_VLANTags.h | 175 |
| 5.32 source/include/FreeRTOSConfigDefaults.h File Reference | 176 |
| 5.32.1 Macro Definition Documentation | 177 |
| 5.32.1.1 tsconfigCONTROLLER_HAS_DYNAMIC_PRIO | 177 |
| 5.32.1.2 tsconfigCONTROLLER_MAX_EVENT_WAIT | 177 |
| 5.32.1.3 tsconfigDEFAULT_QUEUE_TIMEOUT | 177 |
| 5.32.1.4 tsconfigDISABLE | 177 |
| 5.32.1.5 tsconfigDUMP_PACKETS | 177 |
| 5.32.1.6 tsconfigENABLE | 178 |
| 5.32.1.7 tsconfigERRQUEUE_LENGTH | 178 |
| 5.32.1.8 tsconfigINCLUDE_QUEUE_EVENT_CALLBACKS | 178 |
| 5.32.1.9 tsconfigMAX_QUEUE_NAME_LEN | 178 |
| 5.32.1.10 tsconfigSOCKET_INSERTS_VLAN_TAGS | 178 |
| 5.32.1.11 tsconfigTSN_CONTROLLER_PRIORITY | 178 |
| 5.32.1.12 tsconfigWRAPPER_INSERTS_VLAN_TAGS | 178 |
| 5.33 FreeRTOSConfigDefaults.h | 179 |
| 5.34 source/modules/BasicSchedulers/BasicSchedulers.c File Reference | 180 |
| 5.34.1 Detailed Description | 181 |
| 5.34.2 Function Documentation | 181 |
| 5.34.2.1 prvPrioritySelect() | 182 |
| 5.34.2.2 pxNetworkNodeCreateFIFO() | 182 |
| 5.34.2.3 pxNetworkNodeCreatePrio() | 182 |

| | |
|--|-----|
| 5.35 source/modules/BasicSchedulers/BasicSchedulers.h File Reference | 183 |
| 5.35.1 Function Documentation | 183 |
| 5.35.1.1 pxNetworkNodeCreateFIFO() | 184 |
| 5.35.1.2 pxNetworkNodeCreatePrio() | 184 |
| 5.35.1.3 pxNetworkNodeCreateRR() | 184 |
| 5.36 BasicSchedulers.h | 184 |
| 5.37 source/modules/CreditBasedScheduler/SchedCBS.c File Reference | 185 |
| 5.37.1 Detailed Description | 185 |
| 5.37.2 Function Documentation | 185 |
| 5.37.2.1 prvCBSReady() | 186 |
| 5.37.2.2 pxNetworkNodeCreateCBS() | 186 |
| 5.38 source/modules/CreditBasedScheduler/SchedCBS.h File Reference | 187 |
| 5.38.1 Macro Definition Documentation | 188 |
| 5.38.1.1 netschedCBS_DEFAULT_BANDWIDTH | 188 |
| 5.38.1.2 netschedCBS_DEFAULT_MAXCREDIT | 188 |
| 5.38.2 Function Documentation | 188 |
| 5.38.2.1 pxNetworkNodeCreateCBS() | 188 |
| 5.39 SchedCBS.h | 189 |
| 5.40 source/portable/NetworkInterface/Common/NetworkWrapper.c File Reference | 189 |
| 5.40.1 Detailed Description | 190 |
| 5.40.2 Macro Definition Documentation | 190 |
| 5.40.2.1 pxFillInterfaceDescriptor | 190 |
| 5.40.2.2 wrapperFIRST_TPID | 191 |
| 5.40.2.3 wrapperSECOND_TPID | 191 |
| 5.40.2.4 xGetPhyLinkStatus | 191 |
| 5.40.2.5 xNetworkInterfaceInitialise | 191 |
| 5.40.2.6 xNetworkInterfaceOutput | 191 |
| 5.40.2.7 xSendEventStructToIPTask | 191 |
| 5.40.3 Function Documentation | 191 |
| 5.40.3.1 prvAncillaryMsgControlFillForRx() | 192 |
| 5.40.3.2 prvAncillaryMsgControlFillForTx() | 192 |
| 5.40.3.3 prvDumpPacket() | 193 |
| 5.40.3.4 prvHandleReceive() | 193 |
| 5.40.3.5 prvInsertVLANTag() | 194 |
| 5.40.3.6 prvStripVLANTag() | 194 |
| 5.40.3.7 pxTSN_FillInterfaceDescriptor() | 195 |
| 5.40.3.8 vNetworkQueueInit() | 195 |
| 5.40.3.9 vRetrieveHardwareTimestamp() | 196 |
| 5.40.3.10 vTimebaseInit() | 196 |
| 5.40.3.11 xGetPhyLinkStatus() | 197 |
| 5.40.3.12 xNetworkInterfaceInitialise() | 197 |
| 5.40.3.13 xNetworkInterfaceOutput() | 197 |

| | |
|---|------------|
| 5.40.3.14 xSendEventStructToTSNController() | 198 |
| 5.40.3.15 xTSN_GetPhyLinkStatus() | 198 |
| 5.40.3.16 xTSN_NetworkInterfaceInitialise() | 199 |
| 5.40.3.17 xTSN_NetworkInterfaceOutput() | 199 |
| 5.40.4 Variable Documentation | 200 |
| 5.40.4.1 xNetworkWrapperInitialised | 200 |
| 5.41 source/portable/NetworkInterface/include/NetworkWrapper.h File Reference | 200 |
| 5.41.1 Typedef Documentation | 202 |
| 5.41.1.1 NetworkInterfaceConfig_t | 202 |
| 5.41.2 Function Documentation | 202 |
| 5.41.2.1 pxTSN_FillInterfaceDescriptor() | 202 |
| 5.41.2.2 vRetrieveHardwareTimestamp() | 203 |
| 5.41.2.3 xMAC_GetPhyLinkStatus() | 203 |
| 5.41.2.4 xMAC_NetworkInterfaceInitialise() | 203 |
| 5.41.2.5 xMAC_NetworkInterfaceOutput() | 204 |
| 5.41.2.6 xSendEventStructToTSNController() | 204 |
| 5.41.2.7 xTSN_GetPhyLinkStatus() | 205 |
| 5.41.2.8 xTSN_NetworkInterfaceInitialise() | 206 |
| 5.41.2.9 xTSN_NetworkInterfaceOutput() | 206 |
| 5.42 NetworkWrapper.h | 207 |
| Index | 209 |

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 possible 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 `FreeRTOS_TSN_ConfigDefaults.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:

| | |
|---|----|
| cmsghdr | 7 |
| freertos_scm_timestamping | 8 |
| freertos_timespec | |
| FreeRTOS implementation of a timespec | 8 |
| iovec | 9 |
| msghdr | 10 |
| sock_extended_err | 11 |
| struct | 13 |
| xNETQUEUE | |
| A network queue structure, a leaf in the network scheduler tree | 14 |
| xNETQUEUE_ITEM | |
| The structure used in the network scheduler queues | 16 |
| xNETQUEUE_NODE | |
| This is the structure the stores the nodes of the network scheduler | 17 |
| xNETWORK_INTERFACE_CONFIG | 19 |
| xQUEUE_LIST | |
| A list of network queue pointer | 20 |
| xSCHEDULER_CBS | 21 |
| xSCHEDULER_FIFO | 22 |
| xSCHEDULER_GENERIC | |
| A generic structure for implementing a scheduler | 23 |
| xSCHEDULER_PRIO | 24 |
| xSCHEDULER_RR | 25 |
| xTIMEBASE | 26 |
| xTSN_SOCKET | 27 |
| xVLAN_TAG | 29 |

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

| | |
|---|-----|
| source/ FreeRTOS_TSN_Ancillary.c | |
| Implementation of ancillary message functions for FreeRTOS+TCP | 31 |
| source/ FreeRTOS_TSN_Controller.c | |
| FreeRTOS TSN Controller implementation | 38 |
| source/ FreeRTOS_TSN_DS.c | |
| FreeRTOS TSN Compatibility Layer - Data Structures | 45 |
| source/ FreeRTOS_TSN_NetworkScheduler.c | |
| This file contains the implementation of the network scheduler for FreeRTOS TSN Compatibility Layer | 48 |
| source/ FreeRTOS_TSN_NetworkSchedulerBlock.c | |
| Implementation of the FreeRTOS TSN Network Scheduler Block | 55 |
| source/ FreeRTOS_TSN_NetworkSchedulerQueue.c | |
| Implementation of the FreeRTOS TSN Network Scheduler Queue module | 61 |
| source/ FreeRTOS_TSN_Sockets.c | |
| FreeRTOS TSN Compatibility Layer - Socket Functions | 64 |
| source/ FreeRTOS_TSN_Timebase.c | |
| Implementation of the FreeRTOS TSN Timebase module | 74 |
| source/ FreeRTOS_TSN_Timestamp.c | |
| Implementation of the timestamping features | 80 |
| source/ FreeRTOS_TSN_VLANTags.c | |
| Implementation of functions for handling VLAN tags in FreeRTOS TSN Compatibility Layer . . | 82 |
| source/include/ FreeRTOS_TSN_Ancillary.h | 98 |
| source/include/ FreeRTOS_TSN_Controller.h | 109 |
| source/include/ FreeRTOS_TSN_DS.h | 113 |
| source/include/ FreeRTOS_TSN_NetworkScheduler.h | 118 |
| source/include/ FreeRTOS_TSN_NetworkSchedulerBlock.h | 125 |
| source/include/ FreeRTOS_TSN_NetworkSchedulerQueue.h | 131 |
| source/include/ FreeRTOS_TSN_Sockets.h | 136 |
| source/include/ FreeRTOS_TSN_Timebase.h | 148 |
| source/include/ FreeRTOS_TSN_Timestamp.h | 157 |
| source/include/ FreeRTOS_TSN_VLANTags.h | 160 |
| source/include/ FreeRTOS_TSN_ConfigDefaults.h | 176 |
| source/modules/BasicSchedulers/ BasicSchedulers.c | |
| Implementation of some basic schedulers | 180 |
| source/modules/BasicSchedulers/ BasicSchedulers.h | 183 |

| | |
|--|-----|
| source/modules/CreditBasedScheduler/ SchedCBS.c | |
| Implementation of a Credit Based Scheduler | 185 |
| source/modules/CreditBasedScheduler/ SchedCBS.h | 187 |
| source/portable/NetworkInterface/Common/ NetworkWrapper.c | |
| A wrapper for the original NetworkInterface.c | 189 |
| source/portable/NetworkInterface/include/ NetworkWrapper.h | 200 |

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
```

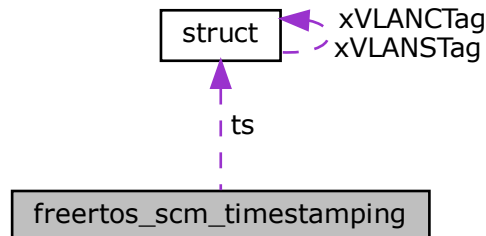
The documentation for this struct was generated from the following file:

- 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
```

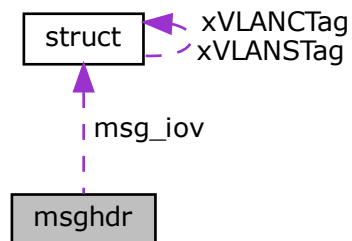
The documentation for this struct was generated from the following file:

- [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
```

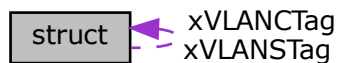
The documentation for this struct was generated from the following file:

- [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 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 add-on. 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 theory 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:

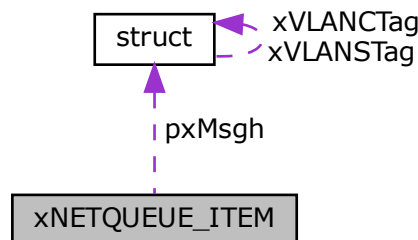
- source/include/FreeRTOS_TSN_NetworkSchedulerQueue.h

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 * pxMsggh
- 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 pxMsggh 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 == pxMsggh
pxMsggh->msg_iov[ 0 ].iov_base == pucOriginalEtherBuffer

```


4.9.2 Field Documentation

4.9.2.1 eEventType

```
eIPEvent_t xNETQUEUE_ITEM::eEventType
```

Specifies whether this packet is a transmission or reception

4.9.2.2 pxBuf

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

4.9.2.3 pxMsg

```
struct msghdr* xNETQUEUE_ITEM::pxMsg
```

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:

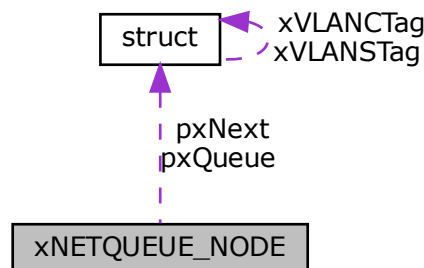
- source/include/[FreeRTOS_TSN_NetworkSchedulerQueue.h](#)

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`, then 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:

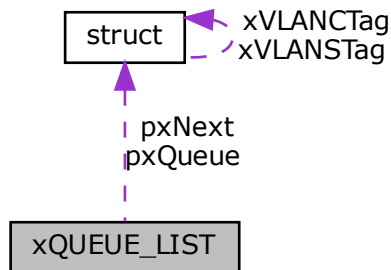
- [source/portable/NetworkInterface/include/NetworkWrapper.h](#)

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 xNETQUEUEUE * pxQueue`
- `struct xQUEUE_LIST * pNext`

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 pNext

```
struct xQUEUE_LIST* xQUEUE_LIST::pNext
```

4.12.2.2 pxQueue

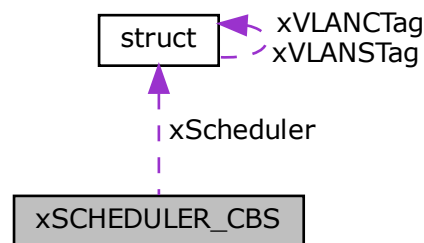
```
struct xNETQUEUE* xQUEUE_LIST::pxQueue
```

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

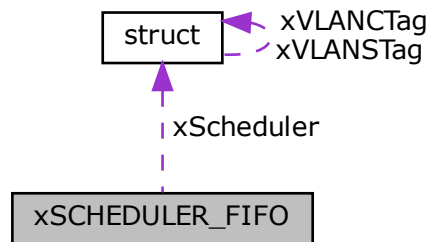
```
struct xSCHEDULER_GENERIC xSCHEDULER_CBS::xScheduler
```

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:

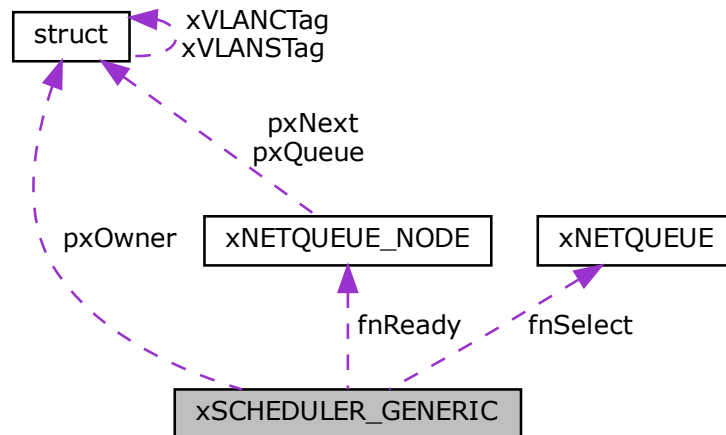
- [source/modules/BasicSchedulers/BasicSchedulers.c](#)

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 intended to be used as is, but should be implemented inside a network scheduler as first member. The size is the length of the specialized version of this struct, which also takes into consideration data in 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`

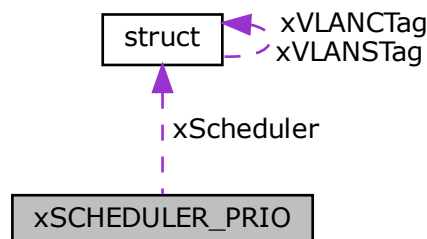
The total length of this structure, counting also the flexible members

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

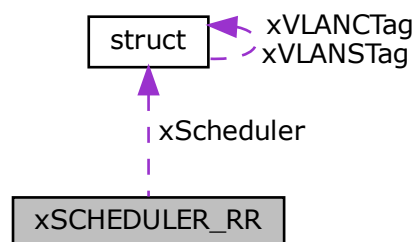
```
struct xSCHEDULER_GENERIC xSCHEDULER_PRIO::xScheduler
```

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
```

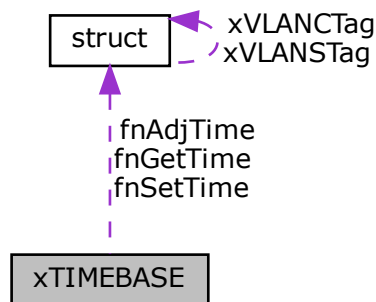
The documentation for this struct was generated from the following file:

- [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` `ucDSCClass`
- `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

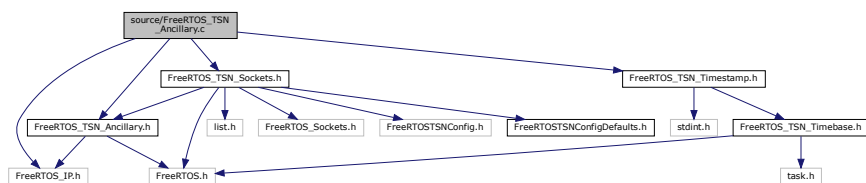
File Documentation

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 *pxMsggh)
Frees the memory allocated for an ancillary message.
- void vAncillaryMsgFreeAll (struct msghdr *pxMsggh)
Frees a msghdr.
- BaseType_t xAncillaryMsgFillName (struct msghdr *pxMsggh, 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 **ppvDataVec, 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 ( )
```

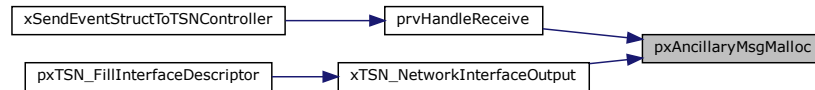
Allocates memory for a new msghdr structure.

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 * pxMsggh )
```

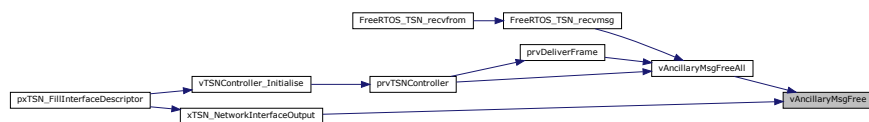
Frees the memory allocated for an ancillary message.

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

Parameters

| | |
|----------------|--|
| <i>pxMsggh</i> | Pointer to the msgghdr structure representing the ancillary message. |
|----------------|--|

Here is the caller graph for this function:

**5.2.2.4 vAncillaryMsgFreeAll()**

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

Frees a msgghdr.

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

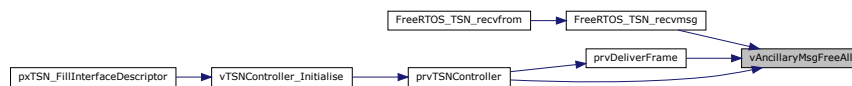
Parameters

| | |
|---------------|---------------------------|
| <i>pxMsgH</i> | 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

| | |
|---------------|---|
| <i>pxMsgH</i> | Pointer to the <code>msghdr</code> 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

| | |
|----------------|-----------------------------------|
| <i>pxMsggh</i> | Pointer to the msgghdr structure. |
|----------------|-----------------------------------|

5.2.2.7 vAncillaryMsgFreePayload()

```
void vAncillaryMsgFreePayload (
    struct msgghdr * pxMsggh )
```

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

| | |
|----------------|--|
| <i>pxMsggh</i> | Pointer to the msgghdr structure representing the ancillary message. |
|----------------|--|

5.2.2.8 xAncillaryMsgControlFill()

```
BaseType_t xAncillaryMsgControlFill (
    struct msgghdr * pxMsggh,
    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.

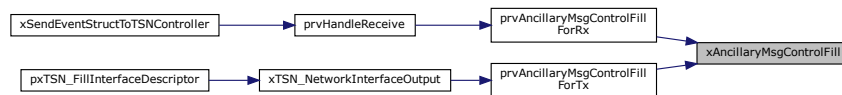
Parameters

| | |
|----------------------|---|
| <i>pxMsggh</i> | A pointer to the msgghdr structure representing the message header. |
| <i>pxCmsgVec</i> | A pointer to the cmsghdr structure representing the control message vector. |
| <i>ppvDataVec</i> | An array of void pointers representing the data vector. |
| <i>puxDataLenVec</i> | An array of size_t values representing the data length vector. |
| <i>uxNumBuffers</i> | 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 * pxMsggh,
    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

| | |
|-------------------|-----------------------------------|
| <i>pxMsggh</i> | Pointer to the msgghdr structure. |
| <i>pxCmsg</i> | Pointer to the cmsghdr structure. |
| <i>pvData</i> | Pointer to the data. |
| <i>puxDataLen</i> | Length of the data. |

Returns

The result of the operation.

5.2.2.10 xAncillaryMsgFillName()

```

BaseType_t xAncillaryMsgFillName (
    struct msghdr * pxMsggh,
    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.

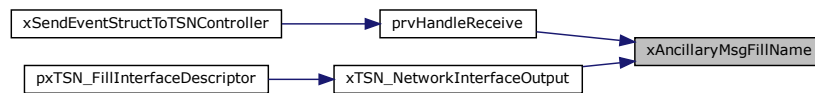
Parameters

| | |
|----------------|---|
| <i>pxMsggh</i> | A pointer to the message header structure. |
| <i>xAddr</i> | A pointer to the IP address to be filled in the name field. |
| <i>usPort</i> | The port number to be filled in the name field. |
| <i>xFamily</i> | 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 * pxMsggh,
    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.

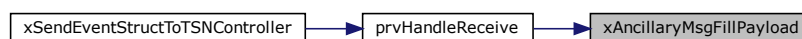
Parameters

| | |
|------------------|---|
| <i>pxMsggh</i> | Pointer to the msghdr structure representing the ancillary message. |
| <i>pucBuffer</i> | Pointer to the buffer containing the payload data. |
| <i>uxLength</i> | 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:



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 )
```

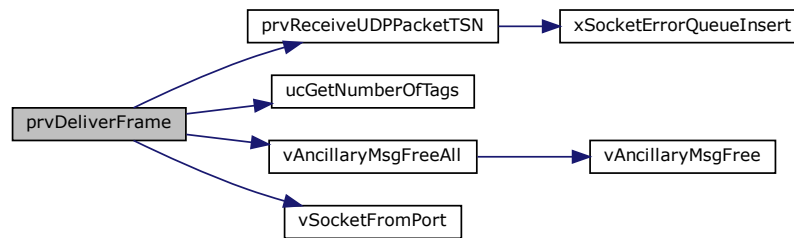
Function to deliver a network frame to the appropriate socket.

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

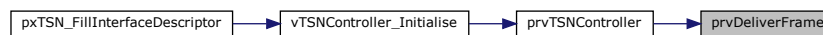
Parameters

| | | |
|----|---------------|--|
| in | <i>pxItem</i> | 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.

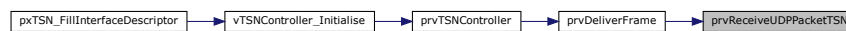
Parameters

| | |
|--------------------|--|
| <i>pxItem</i> | Pointer to the <code>NetworkQueueItem_t</code> structure containing the received packet. |
| <i>xTSNSocket</i> | The TSN socket to receive the packet for. |
| <i>xBaseSocket</i> | 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]
```

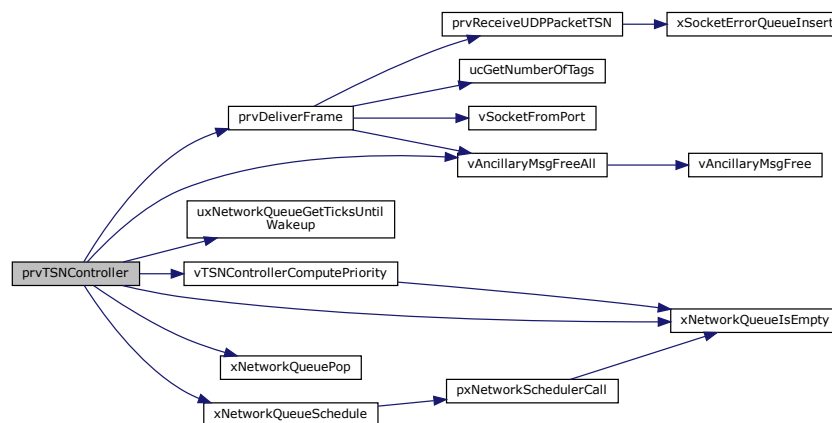
TSN Controller task function.

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.

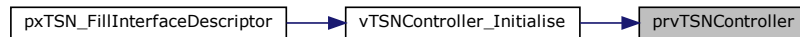
Parameters

| | | |
|----|---------------------|---|
| in | <i>pvParameters</i> | Pointer to the task parameters (not used) |
|----|---------------------|---|

Here is the call graph for this function:



Here is the caller graph for this function:

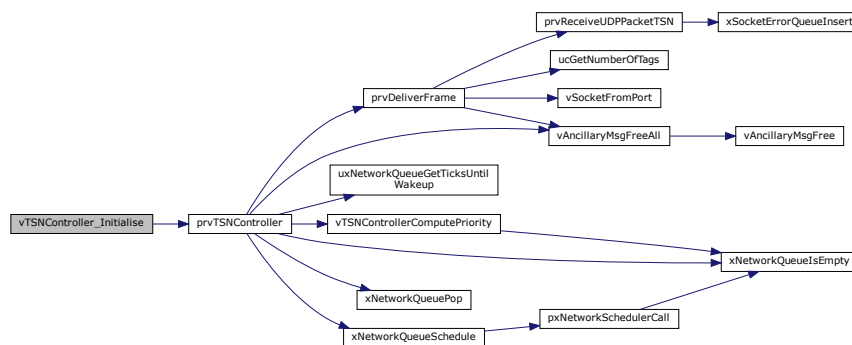


5.3.3.4 vTSNController_Initialise()

```
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()

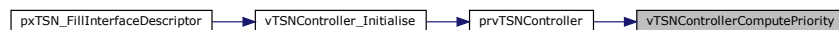
```
void vTSNControllerComputePriority (
    void )
```

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 )
```

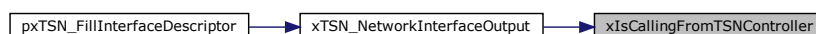
Function to check if the caller task is the TSN Controller task.

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 ( )
```

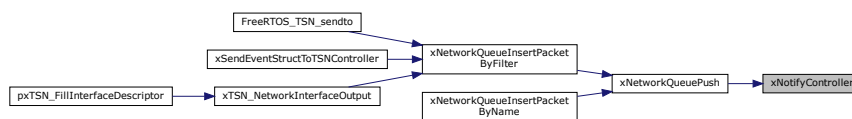
Function to notify the TSN Controller task.

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.

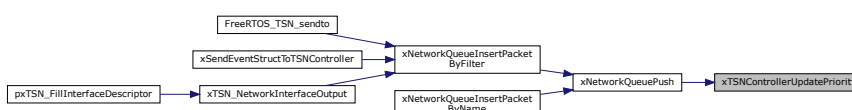
Parameters

| | | |
|----|-------------------|--|
| in | <i>uxPriority</i> | 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]
```

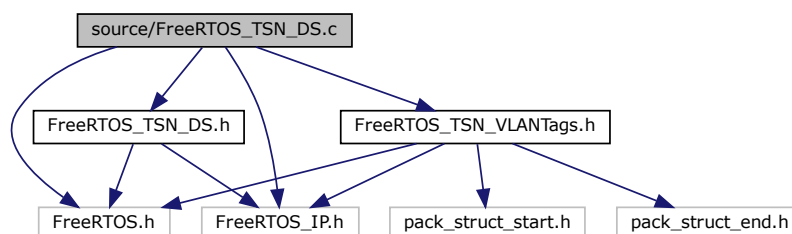
5.3.4.2 xTSNControllerHandle

```
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 [ucDSCClassGet](#) (NetworkBufferDescriptor_t *pxBuf)
Retrieves the DiffServ class from the given network buffer.
- BaseType_t [xDSCClassSet](#) (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 )
```

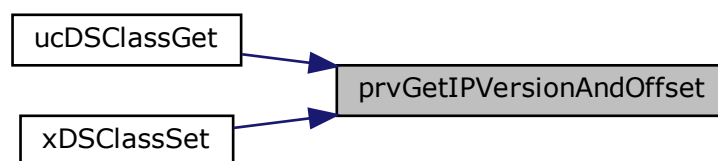
Retrieves the IP version and offset from the given network buffer.

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

Parameters

| | | |
|-----|---------------------|----------------------------------|
| in | <i>pxBuf</i> | The network buffer descriptor. |
| out | <i>pusIPVersion</i> | Pointer to store the IP version. |
| out | <i>pulOffset</i> | Pointer to store the offset. |

Here is the caller graph for this function:



5.4.2.2 ucDSCClassGet()

```
uint8_t ucDSCClassGet (
    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.

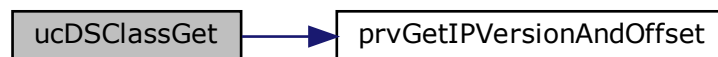
Parameters

| | | |
|----|--------------|--------------------------------|
| in | <i>pxBuf</i> | 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.

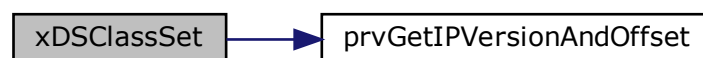
Parameters

| | | |
|----|----------------|----------------------------------|
| in | <i>pxBuf</i> | The network buffer descriptor. |
| in | <i>ucValue</i> | The DiffServ class value to set. |

Returns

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

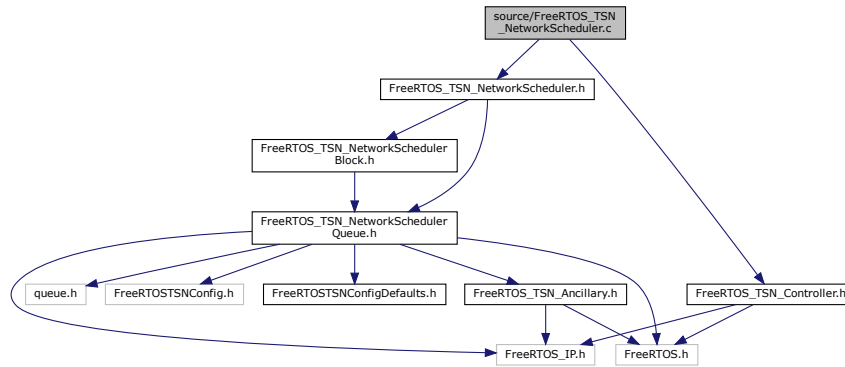
Here is the call graph for this function:



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 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.
- 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.
- 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.

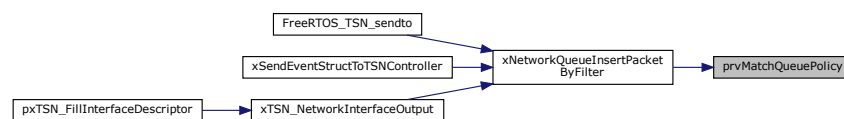
Parameters

| | |
|----------------|-------------------------------------|
| <i>pxItem</i> | The network queue item to match. |
| <i>pxQueue</i> | 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

| | |
|---------------|---------------------------|
| <i>pxItem</i> | 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

| | |
|---------------|---|
| <i>pxNode</i> | 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.

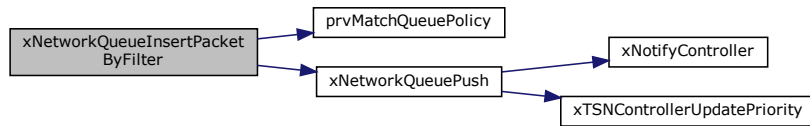
Parameters

| | |
|------------------|--|
| <i>pxItem</i> | The network queue item to insert. |
| <i>uxTimeout</i> | The timeout value for the insertion operation. |

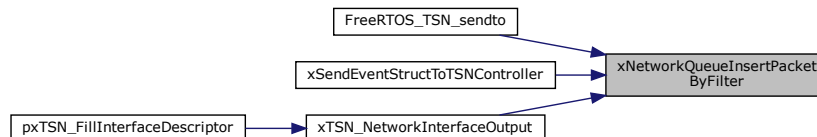
Returns

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.

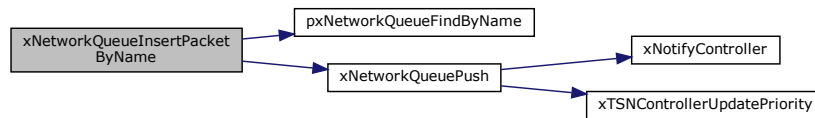
Parameters

| | |
|--------------------|--|
| <i>pxItem</i> | The network queue item to insert. |
| <i>pcQueueName</i> | The name of the network queue to insert into. |
| <i>uxTimeout</i> | 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.

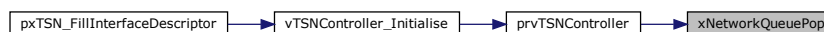
Parameters

| | |
|------------------|--|
| <i>pxQueue</i> | The network queue to pop the item from. |
| <i>pxItem</i> | The network queue item to pop. |
| <i>uxTimeout</i> | 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`.

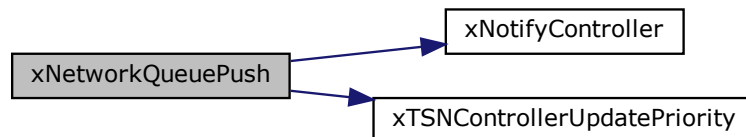
Parameters

| | |
|------------------|---|
| <i>pxQueue</i> | The network queue to push the item into. |
| <i>pxItem</i> | The network queue item to push. |
| <i>uxTimeout</i> | The timeout value for the push operation. |

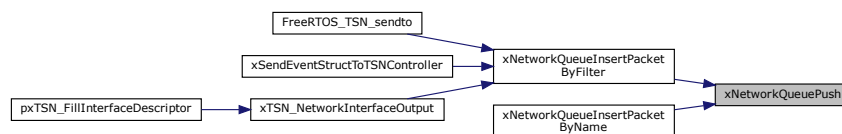
Returns

`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 )
```

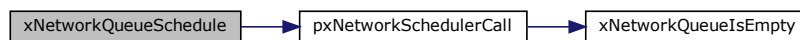
Schedules the network queues and returns the chosen network queue.

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.

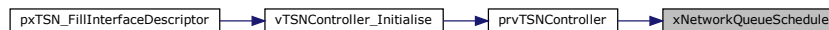
Returns

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
```

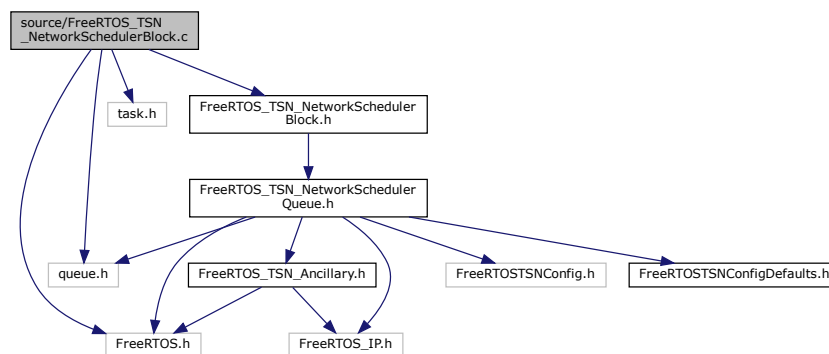
5.5.3.3 uxNumQueues

```
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()

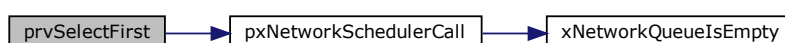
```
NetworkQueue_t * prvSelectFirst (
    NetworkNode_t * pxNode )
```

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

| | |
|---------------|------------------------------|
| <i>pxNode</i> | Pointer to the network node. |
|---------------|------------------------------|

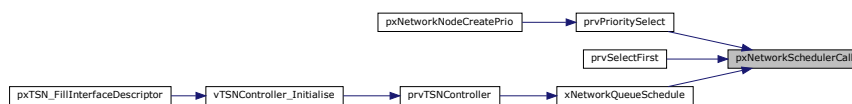
Returns

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.

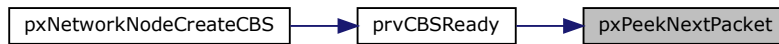
Parameters

| | |
|---------------|------------------------------|
| <i>pxNode</i> | 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 )
```

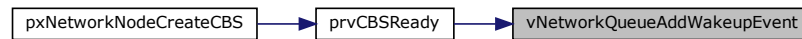
Adds a wakeup event to the network queue.

Although the network scheduler will always periodically 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

| | |
|---------------|--|
| <i>uxTime</i> | 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

| | |
|-------------------|---|
| <i>pxNode</i> | Pointer to the parent network node. |
| <i>pxChild</i> | Pointer to the child network node. |
| <i>uxPosition</i> | 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

| | |
|----------------|-------------------------------|
| <i>pxNode</i> | Pointer to the network node. |
| <i>pxQueue</i> | Pointer to the network queue. |

Returns

pdPASS if the link is successful, pdFAIL otherwise.

5.6.3 Variable Documentation

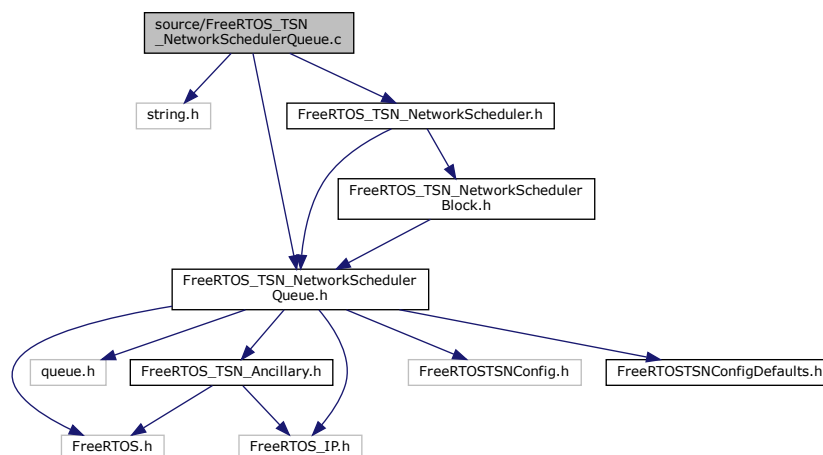
5.6.3.1 uxNextWakeup

```
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

| | |
|--------------|--------------------------------|
| <i>pxBuf</i> | 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

| | |
|--------------|--------------------------------|
| <i>pxBuf</i> | 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

| | |
|----------------|---------------------------------|
| <i>pxQueue</i> | 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.

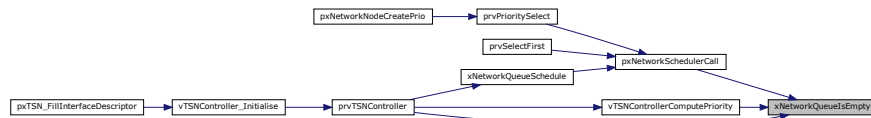
Parameters

| | |
|----------------|---------------------------------|
| <i>pxQueue</i> | 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

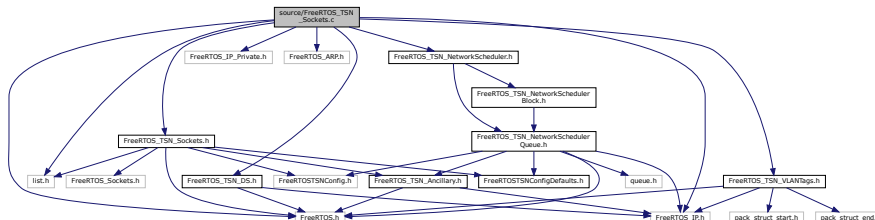
FreeRTOS TSN Compatibility Layer - Socket Functions.

```

#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 *pxMsggh)
- 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 msgghdr *pxMsgghUser, 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

| | |
|-----------------------|--------------------------------------|
| <i>xSocket</i> | The TSN socket to bind. |
| <i>pxAddress</i> | Pointer to the address structure. |
| <i>xAddressLength</i> | 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

| | |
|----------------|------------------------------|
| <i>xSocket</i> | 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.

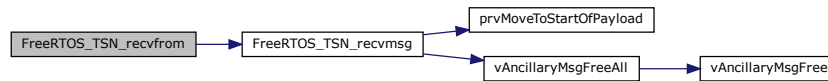
Parameters

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

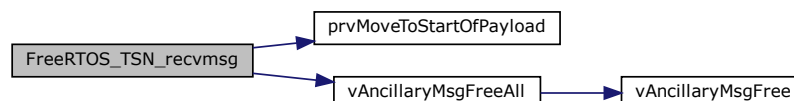
Parameters

| | |
|--------------------|--|
| <i>xSocket</i> | The TSN socket from which to receive the message. |
| <i>pxMsgghUser</i> | Pointer to the <code>msgghdr</code> structure that will hold the received message. |
| <i>xFlags</i> | 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.

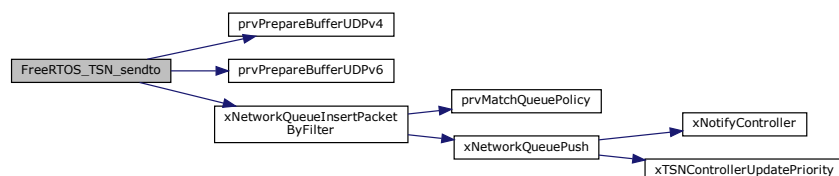
Parameters

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

| | |
|-----------------------|---|
| <i>xSocket</i> | The TSN socket to set options for. |
| <i>lLevel</i> | The level at which the option is defined. |
| <i>lOptionName</i> | The name of the option to set. |
| <i>pvOptionValue</i> | A pointer to the value of the option. |
| <i>uxOptionLength</i> | 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

| | |
|------------------|-----------------------------|
| <i>xDomain</i> | The domain of the socket. |
| <i>xType</i> | The type of the socket. |
| <i>xProtocol</i> | 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 * puvSize )
```

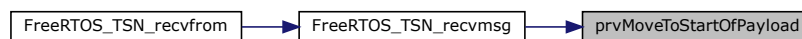
Moves the buffer pointer to the start of the payload and updates the payload size.

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 | <i>ppvBuf</i> | Pointer to the buffer pointer. |
| in, out | <i>puvSize</i> | Pointer to the payload size. |

Here is the caller graph for this function:



5.8.3.9 prvPrepareBufferUDIPv4()

```
BaseType_t prvPrepareBufferUDIPv4 (
    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

| | |
|----------------------------------|--|
| <i>pxSocket</i> | The TSN socket. |
| <i>pxBuf</i> | The network buffer descriptor. |
| <i>xFlags</i> | The flags for the send operation. |
| <i>pxDestinationAddress</i> | The destination address. |
| <i>xDestinationAddressLength</i> | 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

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

Returns

pdFAIL if the buffer preparation fails, pdPASS otherwise.

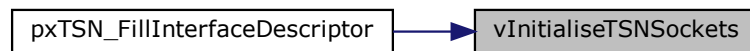
Here is the caller graph for this function:



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 )
```

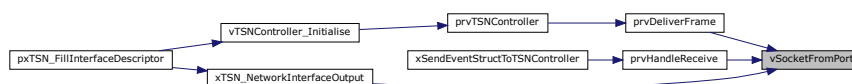
Searches for a socket based on a given search key and retrieves the corresponding base socket and TSN socket.

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

| | |
|---------------------|---|
| <i>xSearchKey</i> | The search key used to find the socket. |
| <i>pxBaseSocket</i> | Pointer to the base socket variable where the retrieved base socket will be stored. |
| <i>pxTSNSocket</i> | Pointer to the TSN socket variable where the retrieved TSN socket will be stored. |

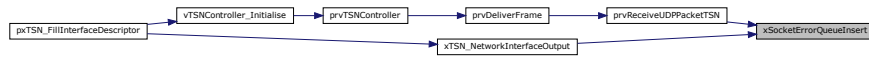
Here is the caller graph for this function:



5.8.3.13 xSocketErrorQueueInsert()

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

Here is the caller graph for this function:



5.8.4 Variable Documentation

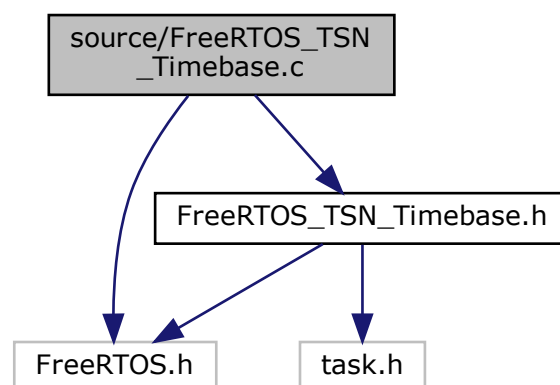
5.8.4.1 xTSNBoundUDPSocketList

```
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

| | |
|-----------|--|
| <i>ts</i> | 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

| | |
|-----------|--|
| <i>ts</i> | 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

| | |
|-------------------|---------------------------------|
| <i>pxTimebase</i> | 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

| | |
|--------------|---|
| <i>pxOp1</i> | Pointer to the first timespec structure. |
| <i>pxOp2</i> | 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

| | |
|--------------|--|
| <i>pxOut</i> | Pointer to the timespec structure to store the result. |
| <i>pxOp1</i> | Pointer to the first timespec structure. |
| <i>pxOp2</i> | 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

| | |
|--------------|--|
| <i>pxOut</i> | Pointer to the timespec structure to store the result. |
| <i>pxOp1</i> | Pointer to the timespec structure to be divided. |
| <i>xOp2</i> | 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

| | |
|--------------|--|
| <i>pxOut</i> | Pointer to the timespec structure to store the result. |
| <i>pxOp1</i> | Pointer to the first timespec structure. |
| <i>pxOp2</i> | 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

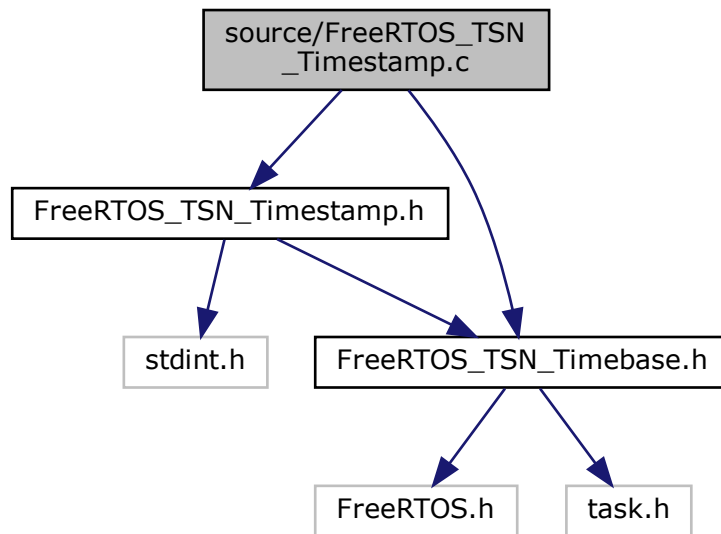
```
eTimebaseState_t xTimebaseState = eTimebaseNotInitialised [static]
```

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 )
```

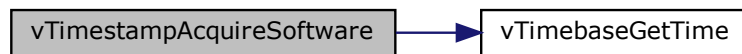
Acquires the software timestamp.

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

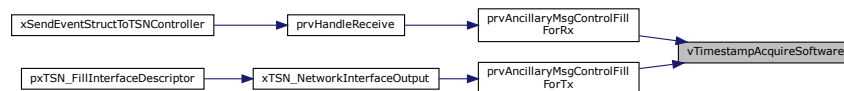
Parameters

| | |
|-----------|---|
| <i>ts</i> | Pointer to the freertos_timespec structure where the acquired timestamp will be stored. |
|-----------|---|

Here is the call graph for this function:



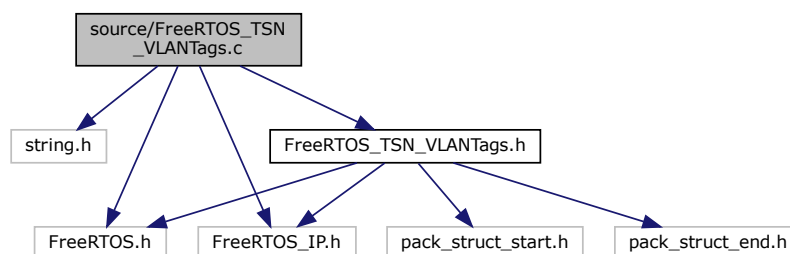
Here is the caller graph for this function:



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.

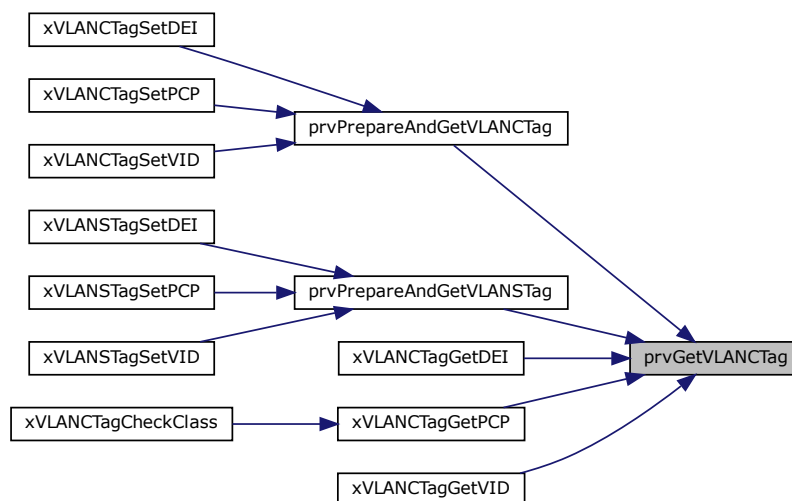
Parameters

| | |
|------------------|--|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>ucNumTags</i> | 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.

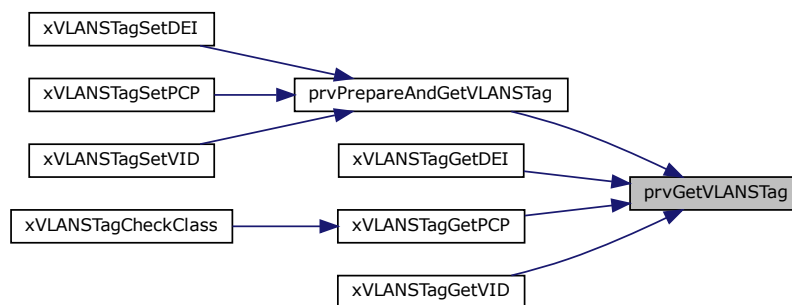
Parameters

| | |
|------------------|--|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>ucNumTags</i> | 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.

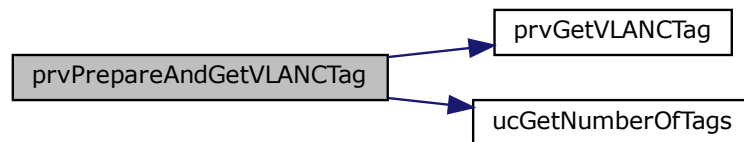
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
|--------------|---|

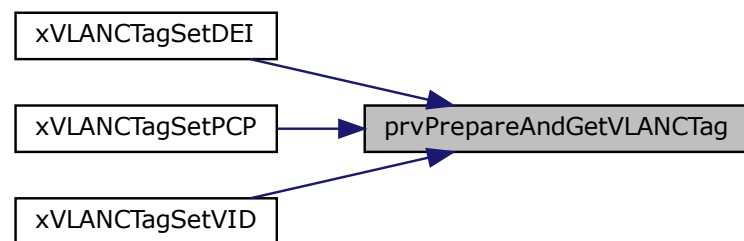
Returns

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.

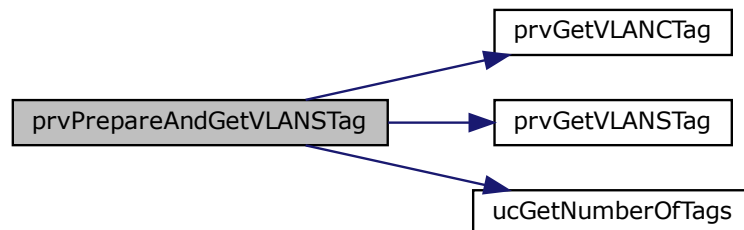
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
|--------------|---|

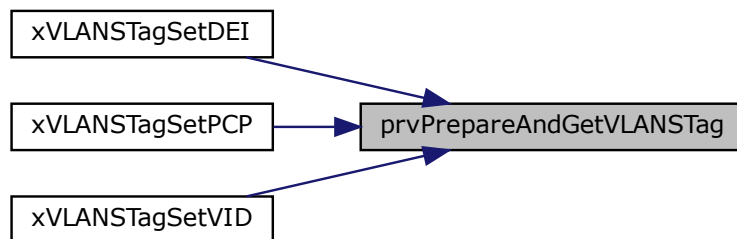
Returns

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

| | |
|--------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
|--------------|---|

Returns

pdTRUE if the PCP value matches, pdFALSE otherwise.

Here is the call graph for this function:

**5.11.2.7 xVLANTagGetDEI()**

```
BaseType_t xVLANTagGetDEI (
    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.

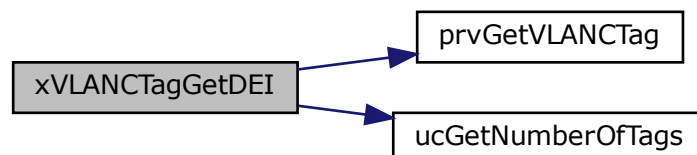
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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.

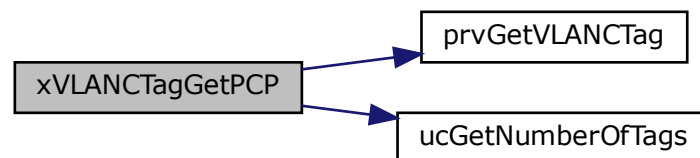
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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.

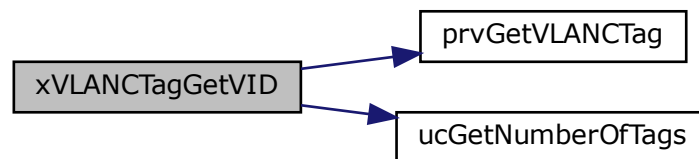
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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.

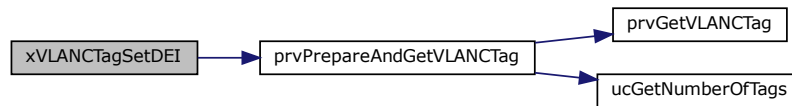
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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 xVLANTagSetPCP()**

```

BaseType_t xVLANTagSetPCP (
    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.

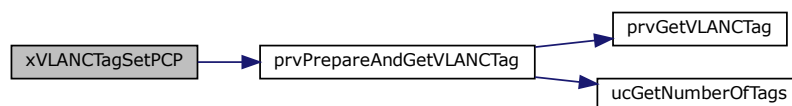
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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 xVLANTagSetVID()

```
BaseType_t xVLANTagSetVID (
    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.

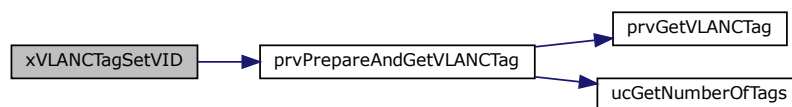
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xClass</i> | 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.

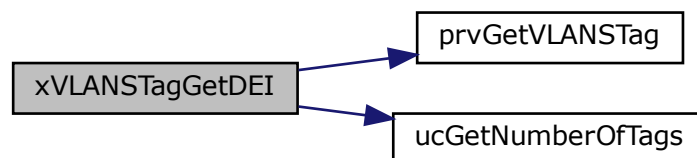
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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.

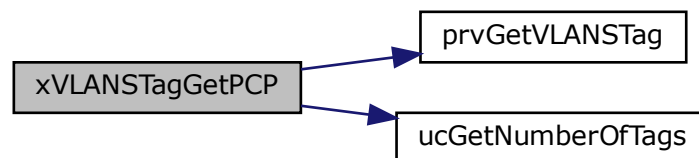
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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.

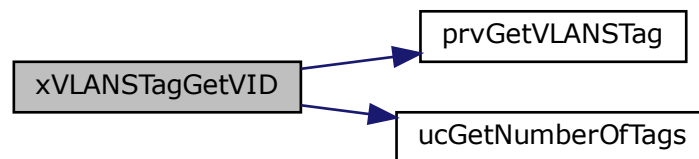
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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.

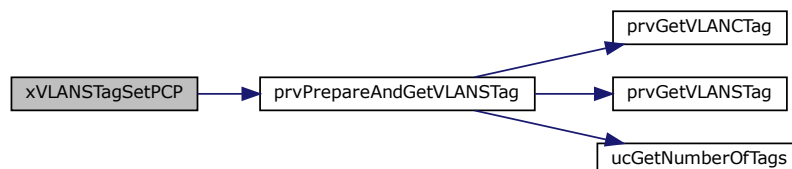
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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.

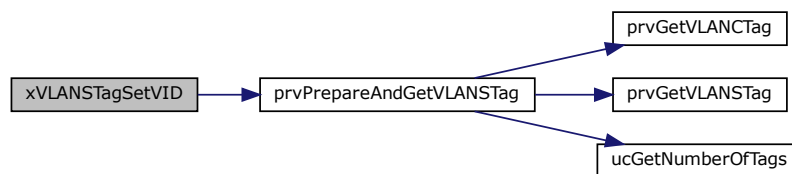
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | The VID value to set. |

Returns

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

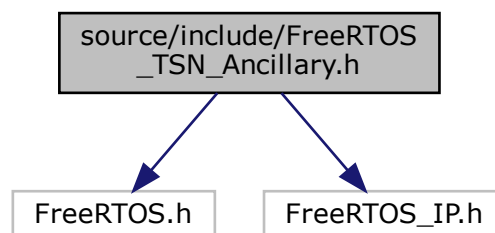
Here is the call graph for this function:



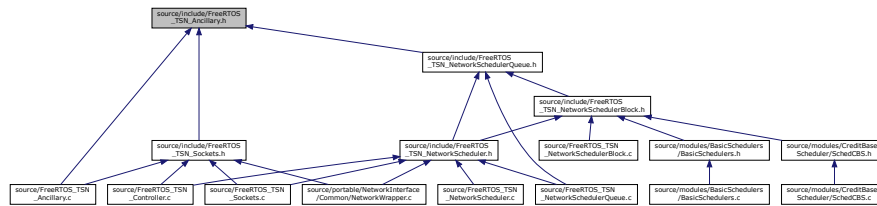
5.12 source/include/FreeRTOS_TSN_Ancillary.h File Reference

```
#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, 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 *pxMsggh)
Frees the memory allocated for the name field in the given msghdr structure.
- BaseType_t [xAncillaryMsgFillPayload](#) (struct msghdr *pxMsggh, uint8_t *pucBuffer, size_t uxLength)
Fills the payload of an ancillary message.
- void [vAncillaryMsgFreePayload](#) (struct msghdr *pxMsggh)
Frees the payload of an ancillary message.
- BaseType_t [xAncillaryMsgControlFill](#) (struct msghdr *pxMsggh, struct cmsghdr *pxCmsgVec, void **pvDataVec, size_t *puxDataLenVec, size_t uxNumBuffers)
Fills the ancillary message control structure with data.
- BaseType_t [xAncillaryMsgControlFillSingle](#) (struct msghdr *pxMsggh, struct cmsghdr *pxCmsg, void *pvData, size_t puxDataLen)
- void [vAncillaryMsgFreeControl](#) (struct msghdr *pxMsggh)
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 * msg )
```

Aligns the size of a control message buffer.

5.12.2.2 `pxAncillaryMsgMalloc()`

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

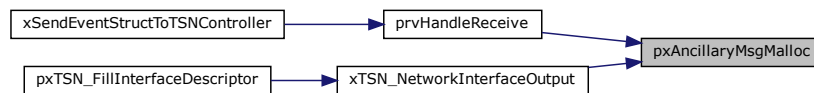
Allocates memory for a new msghdr structure.

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 * pxMsggh )
```

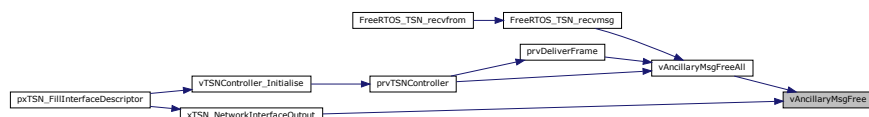
Frees the memory allocated for an ancillary message.

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

Parameters

| | |
|----------------|---|
| <i>pxMsggh</i> | 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.

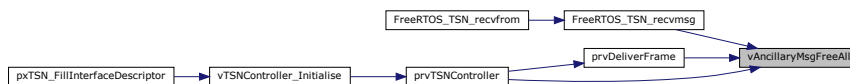
Parameters

| | |
|---------------|---------------------------|
| <i>pxMsgh</i> | 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

| | |
|---------------|----------------------------------|
| <i>pxMsgh</i> | 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

| | |
|---------------|----------------------------------|
| <i>pxMsgh</i> | 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

| | |
|---------------|---|
| <i>pxMsgh</i> | 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.

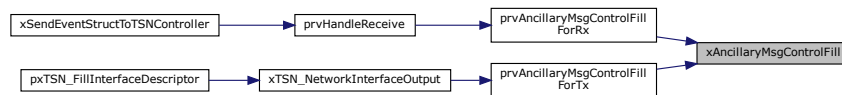
Parameters

| | |
|----------------------|---|
| <i>pxMsgh</i> | A pointer to the msghdr structure representing the message header. |
| <i>pxCmsgVec</i> | A pointer to the cmsghdr structure representing the control message vector. |
| <i>ppvDataVec</i> | An array of void pointers representing the data vector. |
| <i>puxDataLenVec</i> | An array of size_t values representing the data length vector. |
| <i>uxNumBuffers</i> | 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 * pxMsggh,
    struct cmsghdr * pxCmsg,
    void * pvData,
    size_t puxDataLen )
  
```

5.12.2.10 xAncillaryMsgFillName()

```

BaseType_t xAncillaryMsgFillName (
    struct msghdr * pxMsggh,
    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.

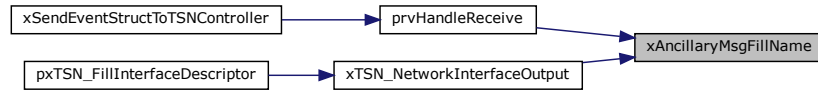
Parameters

| | |
|----------------|---|
| <i>pxMsggh</i> | A pointer to the message header structure. |
| <i>xAddr</i> | A pointer to the IP address to be filled in the name field. |
| <i>usPort</i> | The port number to be filled in the name field. |
| <i>xFamily</i> | 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 * pxMsggh,
    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.

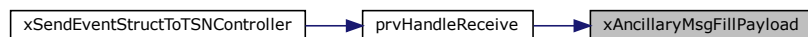
Parameters

| | |
|------------------|--|
| <i>pxMsggh</i> | Pointer to the msgghdr structure representing the ancillary message. |
| <i>pucBuffer</i> | Pointer to the buffer containing the payload data. |
| <i>uxLength</i> | 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 msg_hdr
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 MSG_ALIGN( len ) ( ( ( len ) + sizeof( long ) - 1 ) & ~( sizeof( long ) - 1 ) )
60
61 #define MSG_DATA( cmsg ) ( ( void * ) ( ( char * ) ( cmsg ) + MSG_ALIGN( sizeof( struct cmsghdr ) ) ) )
62
63 #define MSG_SPACE( len ) ( MSG_ALIGN( sizeof( struct cmsghdr ) ) + MSG_ALIGN( len ) )
64
65 #define MSG_LEN( len ) ( MSG_ALIGN( sizeof( struct cmsghdr ) ) + ( len ) )
66
67 #define __MSG_FIRSTHDR( ctl, len ) \
68 ( ( len ) >= sizeof( struct cmsghdr ) ? \
69 ( struct cmsghdr * ) ( ctl ) : \
70 ( struct cmsghdr * ) NULL )
71
72 #define MSG_FIRSTHDR( msg ) __MSG_FIRSTHDR( ( msg )->msg_control, ( msg )->msg_controllen )
73
74 struct cmsghdr * __MSG_NXTHDR( void * ctl,
75 size_t size,
76 struct cmsghdr * cmsg );
77
78 #define MSG_NXTHDR( mhdr, cmsg ) __MSG_NXTHDR( ( mhdr )->msg_control, ( mhdr )->msg_controllen, ( cmsg ) )
79
80
81 struct msg_hdr * pxAncillaryMsgMalloc();
82
83 void vAncillaryMsgFree( struct msg_hdr * pxMsg );
84
85 void vAncillaryMsgFreeAll( struct msg_hdr * pxMsg );
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 */

```

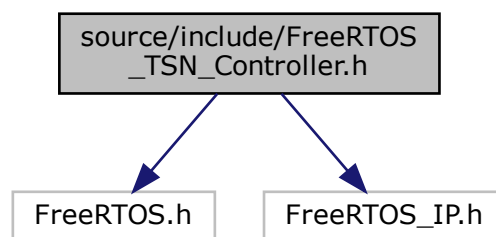
5.14 source/include/FreeRTOS_TSN_Controller.h File Reference

```

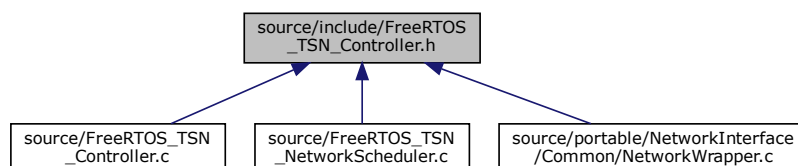
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"

```

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 [xlsCallingFromTSNController](#) (void)
Function to check if the caller task is the TSN Controller task.

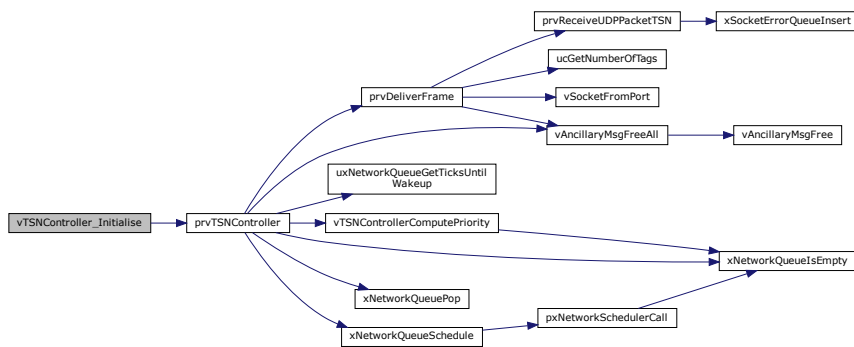
5.14.1 Function Documentation

5.14.1.1 vTSNController_Initialise()

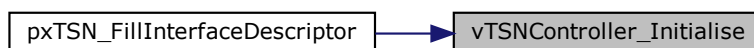
```
void vTSNController_Initialise (
    void )
```

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()

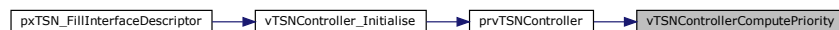
```
void vTSNControllerComputePriority (
    void )
```

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 )
```

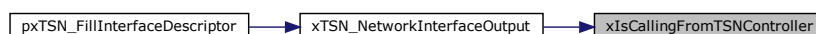
Function to check if the caller task is the TSN Controller task.

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 ( )
```

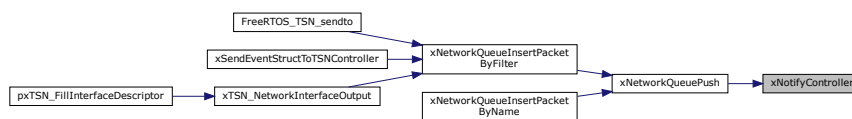
Function to notify the TSN Controller task.

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.

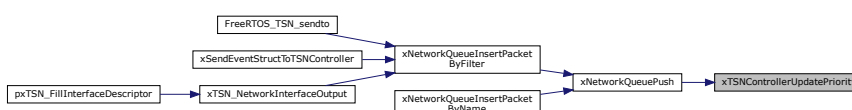
Parameters

| | | |
|----|-------------------|--|
| in | <i>uxPriority</i> | 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 vTSNControllerInitialise( void );
14
15 BaseType_t xIsCallingFromTSNController( void );
16
17 #endif /* ifndef FREERTOS_TSN_CONTROLLER */

```

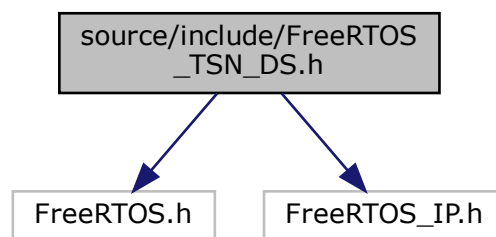
5.16 source/include/FreeRTOS_TSN_DS.h File Reference

```

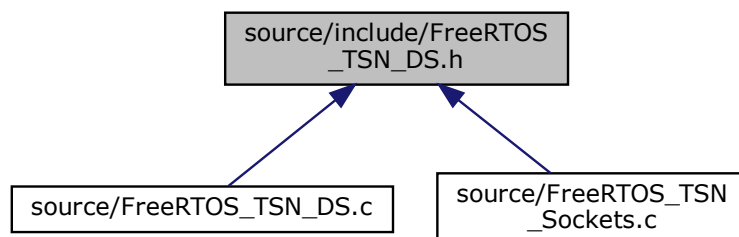
#include "FreeRTOS.h"
#include "FreeRTOS_IP.h"

```

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 *pBuf)`
Retrieves the DiffServ class from the given network buffer.
- `BaseType_t xDSClassSet (NetworkBufferDescriptor_t *pBuf, 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 ) | ( ( pxIPHeader->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.

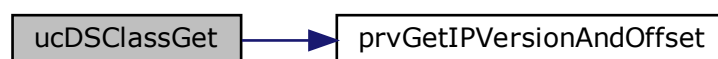
Parameters

| | | |
|----|--------------|--------------------------------|
| in | <i>pxBuf</i> | 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.

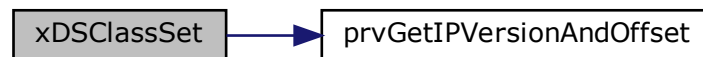
Parameters

| | | |
|----|----------------|----------------------------------|
| in | <i>pxBuf</i> | The network buffer descriptor. |
| in | <i>ucValue</i> | 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 */

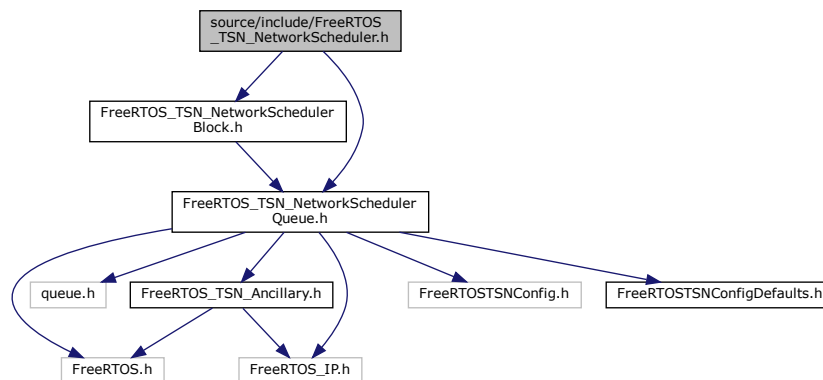
```

5.18 source/include/FreeRTOS_TSN_NetworkScheduler.h File Reference

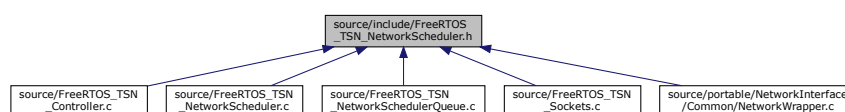
```
#include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
```

```
#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 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.
- 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.
- `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 * pItem )
```

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

| | |
|--------------|---------------------------|
| <i>pItem</i> | The network queue to add. |
|--------------|---------------------------|

5.18.2.4 xNetworkQueueAssignRoot()

```
BaseType_t xNetworkQueueAssignRoot (
    NetworkNode_t * pNode )
```

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

| | |
|--------------|---|
| <i>pNode</i> | 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 * pItem,
    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.

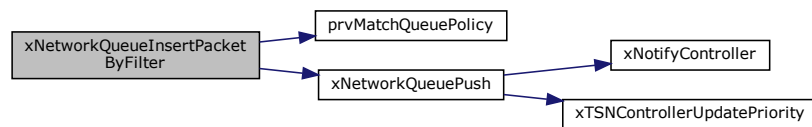
Parameters

| | |
|------------------|--|
| <i>pxItem</i> | The network queue item to insert. |
| <i>uxTimeout</i> | The timeout value for the insertion operation. |

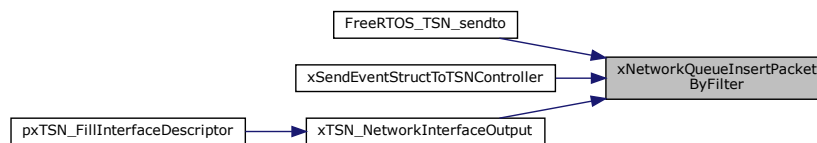
Returns

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.

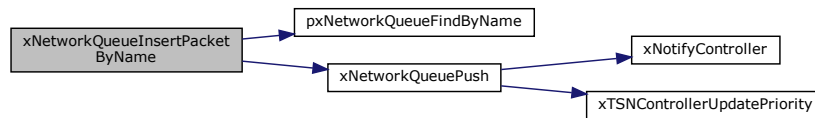
Parameters

| | |
|--------------------|--|
| <i>pxItem</i> | The network queue item to insert. |
| <i>pcQueueName</i> | The name of the network queue to insert into. |
| <i>uxTimeout</i> | 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.

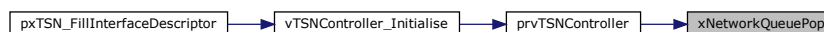
Parameters

| | |
|------------------|--|
| <i>pxQueue</i> | The network queue to pop the item from. |
| <i>pxItem</i> | The network queue item to pop. |
| <i>uxTimeout</i> | 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`.

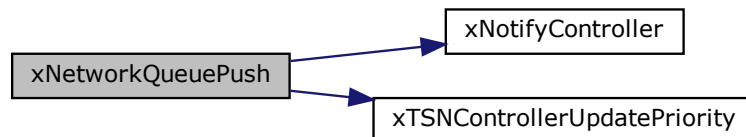
Parameters

| | |
|------------------|---|
| <i>pxQueue</i> | The network queue to push the item into. |
| <i>pxItem</i> | The network queue item to push. |
| <i>uxTimeout</i> | The timeout value for the push operation. |

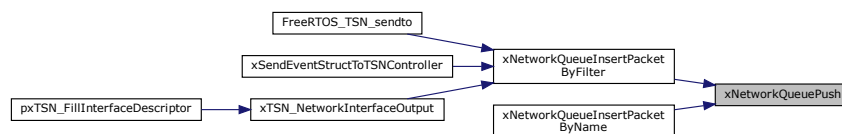
Returns

`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 )
```

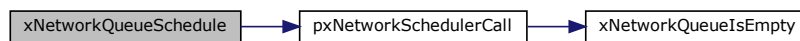
Schedules the network queues and returns the chosen network queue.

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.

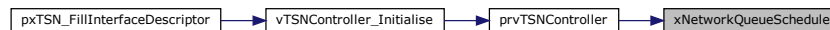
Returns

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
```

```

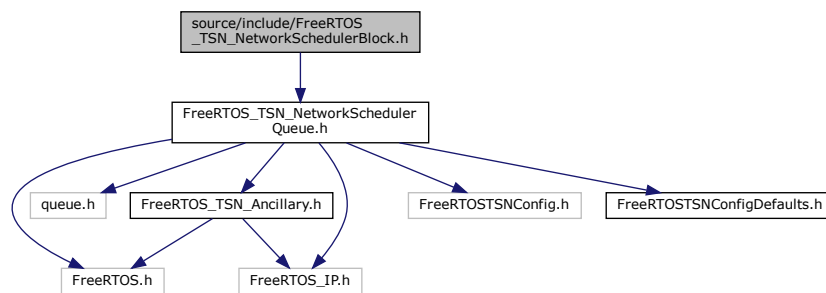
34 NetworkQueue_t * xNetworkQueueSchedule( void );
35
36 BaseType_t xNetworkQueuePush( NetworkQueue_t * pxQueue,
37                               const NetworkQueueItem_t * pxItem,
38                               UBaseType_t uxTimeout );
39
40 BaseType_t xNetworkQueuePop( NetworkQueue_t * pxQueue,
41                              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 */

```

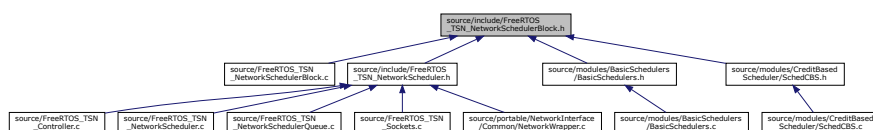
5.20 source/include/FreeRTOS_TSN_NetworkSchedulerBlock.h File Reference

#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 that 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

| | |
|---------------|------------------------------|
| <i>pxNode</i> | Pointer to the network node. |
|---------------|------------------------------|

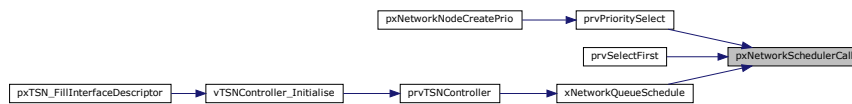
Returns

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.

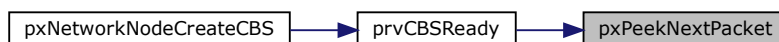
Parameters

| | |
|---------------|------------------------------|
| <i>pxNode</i> | 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 )
```

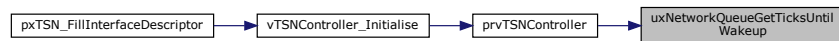
Gets the ticks until the next wakeup event.

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 )
```

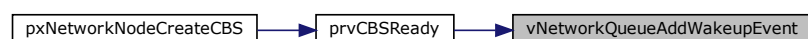
Adds a wakeup event to the network queue.

Although the network scheduler will always periodically 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

| | |
|---------------|--|
| <i>uxTime</i> | 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

| | |
|-------------------|---|
| <i>pxNode</i> | Pointer to the parent network node. |
| <i>pxChild</i> | Pointer to the child network node. |
| <i>uxPosition</i> | 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

| | |
|----------------|-------------------------------|
| <i>pxNode</i> | Pointer to the network node. |
| <i>pxQueue</i> | 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
27 struct xSCHEDULER_GENERIC
28 {
29     {
30         uint16_t usSize;
31         struct xNETQUEUE_NODE * pxOwner;
32         SelectQueueFunction_t fnSelect;
33         ReadyQueueFunction_t fnReady;
34         char ucAttributes[];
35     };
36
37 #if ( configSUPPORT_DYNAMIC_ALLOCATION != 0 )
38     NetworkNode_t * pxNetworkNodeCreate( BaseType_t uxNumChildren );
39
40     void vNetworkNodeRelease( NetworkNode_t * pxNode );
41
42     void * pvNetworkSchedulerGenericCreate( NetworkNode_t * pxNode,
43                                             uint16_t usSize );
44
45     void vNetworkSchedulerGenericRelease( void * pvSched );
46 #endif /* if ( configSUPPORT_DYNAMIC_ALLOCATION != 0 ) */
47
48 BaseType_t xNetworkSchedulerLinkQueue( NetworkNode_t * pxNode,
49                                       NetworkQueue_t * pxQueue );
50
51 BaseType_t xNetworkSchedulerLinkChild( NetworkNode_t * pxNode,
52                                       NetworkNode_t * pxChild,
53                                       size_t uxPosition );
54
55 NetworkQueue_t * pxNetworkSchedulerCall( NetworkNode_t * pxNode );
56
57 NetworkBufferDescriptor_t * pxPeekNextPacket( NetworkNode_t * pxNode );
58
59 TickType_t uxNetworkQueueGetTicksUntilWakeup( void );
60
61 void vNetworkQueueAddWakeupEvent( TickType_t uxTime );
62
63 #define netschedCALL_SELECT_FROM_NODE( pxNode ) \
64 ( ( ( struct xSCHEDULER_GENERIC * ) pxNode->pvScheduler )->fnSelect( pxNode ) )
65
66 #define netschedCALL_READY_FROM_NODE( pxNode ) \
67 ( ( ( struct xSCHEDULER_GENERIC * ) pxNode->pvScheduler )->fnReady( pxNode ) )
68
69 #endif /* FREERTOS_TSN_NETWORK_SCHEDULER_H */

```

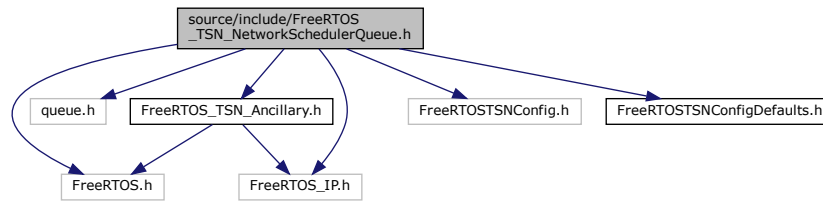
5.22 source/include/FreeRTOS_TSN_NetworkSchedulerQueue.h File Reference

```

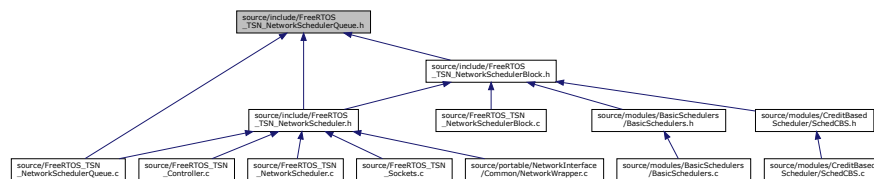
#include "FreeRTOS.h"
#include "queue.h"
#include "FreeRTOS_IP.h"
#include "FreeRTOS_TSN_Ancillary.h"
#include "FreeRTOS_TSN_Config.h"
#include "FreeRTOS_TSN_ConfigDefaults.h"

```

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

- BaseType_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

| | |
|----------------|---------------------------------|
| <i>pxQueue</i> | 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.

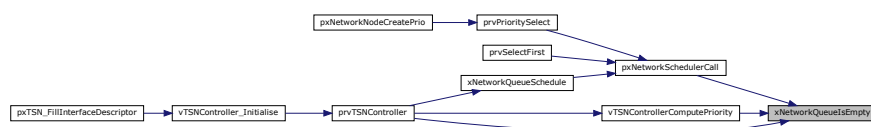
Parameters

| | |
|----------------|---------------------------------|
| <i>pxQueue</i> | 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.](#)

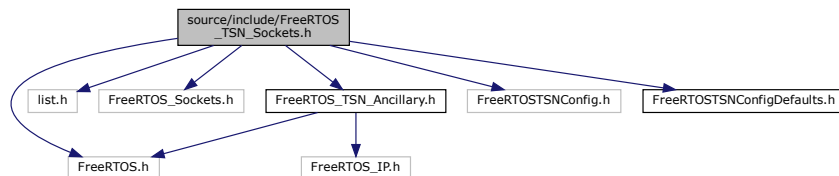
```

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 "FreeRTOS_TSN_Config.h"
12 #include "FreeRTOS_TSN_ConfigDefaults.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 * pxMsg;
58     BaseType_t xReleaseAfterSend;
59 };
60
61 typedef struct xNETQUEUE_ITEM NetworkQueueItem_t;
62
63 struct xNETQUEUE
64 {
65     QueueHandle_t xQueue;
66     UBaseType_t uxIPV;
67     eQueuePolicy_t ePolicy;
68 #if ( tsconfigMAX_QUEUE_NAME_LEN != 0 )
69     char cName[ tsconfigMAX_QUEUE_NAME_LEN ];
70 #endif
71 FilterFunction_t fnFilter;
72 #if ( tsconfigINCLUDE_QUEUE_EVENT_CALLBACKS != tsconfigDISABLE )
73     PacketHandleFunction_t fnOnPop;
74     PacketHandleFunction_t fnOnPush;
75 #endif
76 };
77
78 typedef struct xNETQUEUE NetworkQueue_t;
79
80 #if ( configSUPPORT_DYNAMIC_ALLOCATION != 0 )
81
82     NetworkQueue_t * pxNetworkQueueMalloc();
83
84     NetworkQueue_t * pxNetworkQueueCreate( eQueuePolicy_t ePolicy,
85                                             UBaseType_t uxIPV,
86                                             char * cName,
87                                             FilterFunction_t fnFilter );
88
89     void vNetworkQueueFree( NetworkQueue_t * pxQueue );
90
91     NetworkQueueItem_t * pxNetworkQueueItemMalloc();
92
93     void vNetworkQueueItemFree( NetworkQueueItem_t * pxItem );
94
95 #endif /* if ( configSUPPORT_DYNAMIC_ALLOCATION != 0 ) */
96
97 UBaseType_t uxNetworkQueuePacketsWaiting( NetworkQueue_t * pxQueue );
98
99 BaseType_t xNetworkQueueIsEmpty( NetworkQueue_t * pxQueue );
100
101 #endif /* FREERTOS_TSN_NETWORK_SCHEDULER_QUEUE_H */

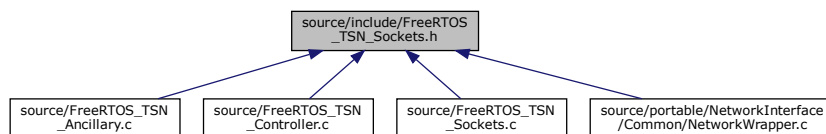
```

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 "FreeRTOS_TSNConfig.h"
#include "FreeRTOS_TSNConfigDefaults.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 ((TSN_Socket_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](#) *pxMsggh)
- [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_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](#) *pxMsgghUser, 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

| | |
|-----------------------|--------------------------------------|
| <i>xSocket</i> | The TSN socket to bind. |
| <i>pxAddress</i> | Pointer to the address structure. |
| <i>xAddressLength</i> | 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

| | |
|----------------|------------------------------|
| <i>xSocket</i> | 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.

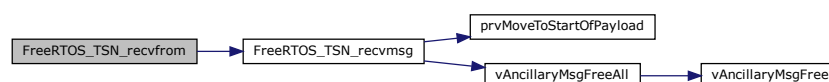
Parameters

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

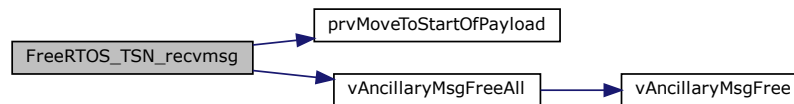
Parameters

| | |
|--------------------|---|
| <i>xSocket</i> | The TSN socket from which to receive the message. |
| <i>pxMsgghUser</i> | Pointer to the <code>msghdr</code> structure that will hold the received message. |
| <i>xFlags</i> | 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.

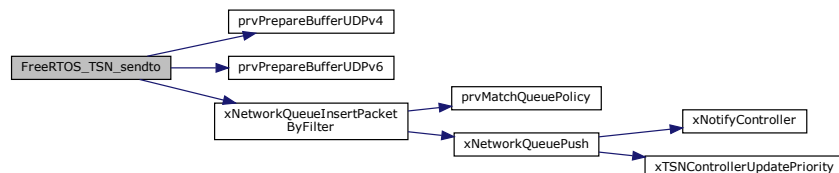
Parameters

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

| | |
|-----------------------|---|
| <i>xSocket</i> | The TSN socket to set options for. |
| <i>lLevel</i> | The level at which the option is defined. |
| <i>lOptionName</i> | The name of the option to set. |
| <i>pvOptionValue</i> | A pointer to the value of the option. |
| <i>uxOptionLength</i> | 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

| | |
|------------------|-----------------------------|
| <i>xDomain</i> | The domain of the socket. |
| <i>xType</i> | The type of the socket. |
| <i>xProtocol</i> | The protocol of the socket. |

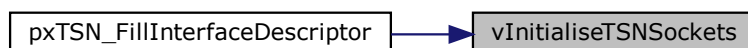
Returns

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 )
```

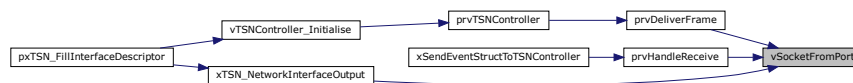
Searches for a socket based on a given search key and retrieves the corresponding base socket and TSN socket.

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

| | |
|---------------------|---|
| <i>xSearchKey</i> | The search key used to find the socket. |
| <i>pxBaseSocket</i> | Pointer to the base socket variable where the retrieved base socket will be stored. |
| <i>pxTSNSocket</i> | Pointer to the TSN socket variable where the retrieved TSN socket will be stored. |

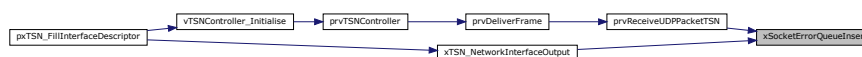
Here is the caller graph for this function:



5.24.4.10 xSocketErrorQueueInsert()

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

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 "FreeRTOS_TSNConfig.h"
12 #include "FreeRTOS_TSNConfigDefaults.h"
13
14 #define FREERTOS_TSN_INVALID_SOCKET      ( ( TSN_Socket_t ) ~0U )
15
16 #if ( tsconfigSOCKET_INSERTS_VLAN_TAGS != tsconfigDISABLE ) /*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;
85     QueueHandle_t xErrQueue;
86     #if ( tsconfigSOCKET_INSERTS_VLAN_TAGS != tsconfigDISABLE )
87         uint8_t ucVLANTagsCount;
88         uint16_t usVLANTagTCI;
89         uint16_t usVLANSTagTCI;
90     #endif
91 };

```

```

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

```

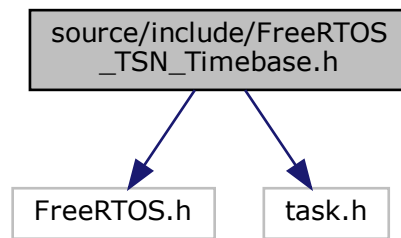
5.26 source/include/FreeRTOS_TSN_Timebase.h File Reference

```

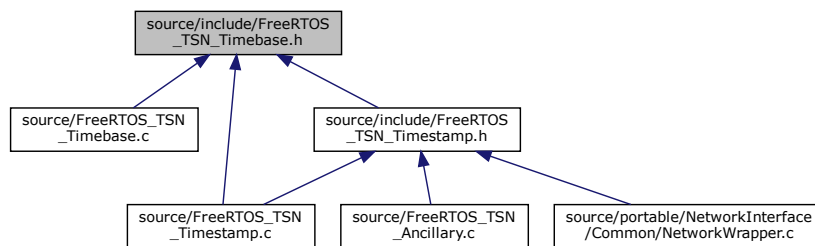
#include "FreeRTOS.h"
#include "task.h"

```

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()

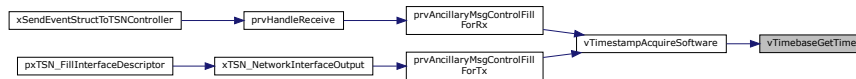
```
void vTimebaseGetTime (
    struct freertos_timespec * ts )
```

Gets the current time of the timebase.

Parameters

| | |
|-----------|--|
| <i>ts</i> | 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

| | |
|-----------|--|
| <i>ts</i> | 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()

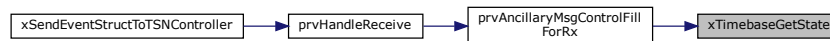
```
BaseType_t xTimebaseGetState (
    void )
```

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

| | |
|-------------------|---------------------------------|
| <i>pxTimebase</i> | 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

| | |
|--------------|---|
| <i>pxOp1</i> | Pointer to the first timespec structure. |
| <i>pxOp2</i> | 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

| | |
|--------------|--|
| <i>pxOut</i> | Pointer to the timespec structure to store the result. |
| <i>pxOp1</i> | Pointer to the first timespec structure. |
| <i>pxOp2</i> | 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

| | |
|--------------|--|
| <i>pxOut</i> | Pointer to the timespec structure to store the result. |
| <i>pxOp1</i> | Pointer to the timespec structure to be divided. |
| <i>xOp2</i> | 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

| | |
|--------------|--|
| <i>pxOut</i> | Pointer to the timespec structure to store the result. |
| <i>pxOp1</i> | Pointer to the first timespec structure. |
| <i>pxOp2</i> | 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 */

```

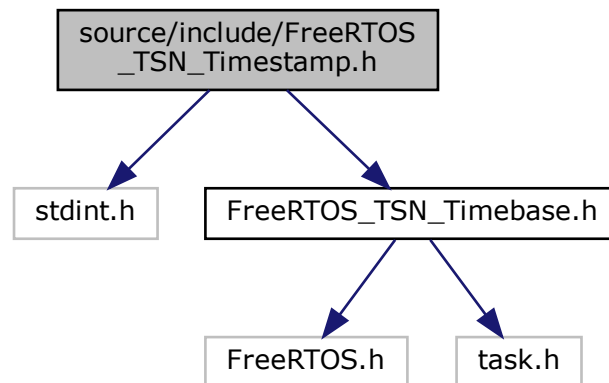
5.28 source/include/FreeRTOS_TSN_Timestamp.h File Reference

```

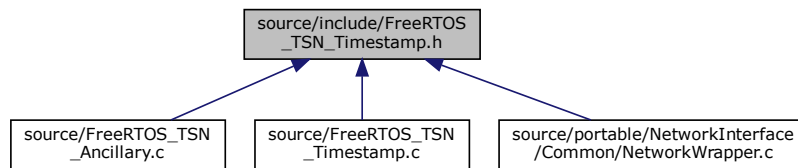
#include <stdint.h>
#include "FreeRTOS_TSN_Timebase.h"

```

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 )
```

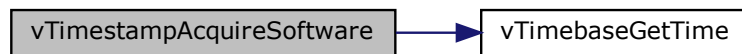
Acquires the software timestamp.

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

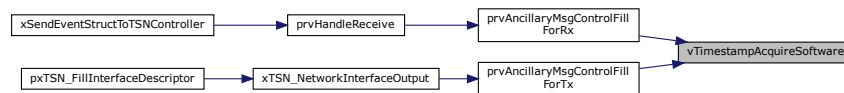
Parameters

| | |
|-----------------|---|
| <code>ts</code> | 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

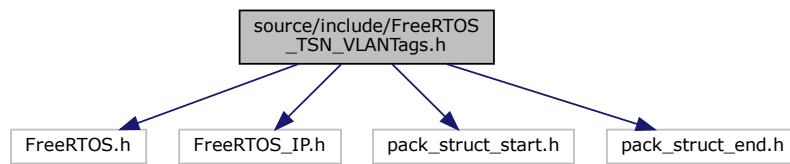
```

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

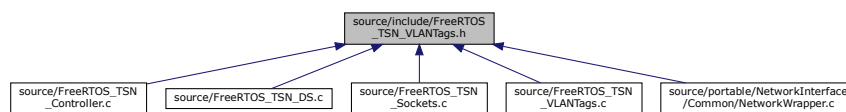


```
#include "pack_struct_end.h"
```

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 & 0xFFFFU ) ) ); \  
} 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 ( 0xFFFFU )
```

5.30.1.21 xVLANTagCheckClass

```
#define xVLANTagCheckClass xVLANTagCheckClass
```

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.

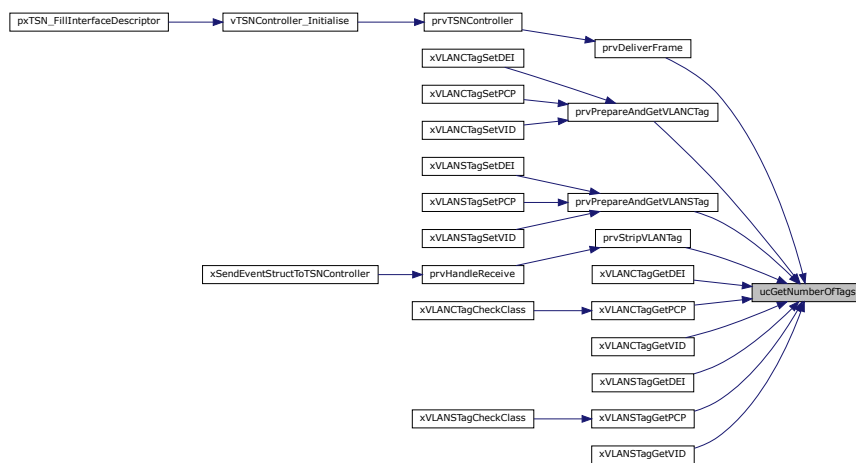
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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.

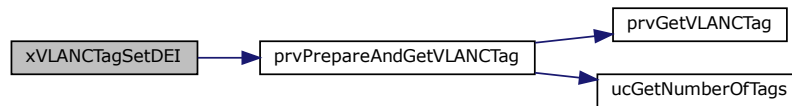
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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 xVLANTagSetPCP()**

```

BaseType_t xVLANTagSetPCP (
    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.

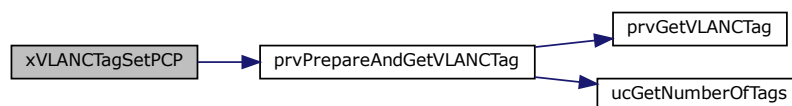
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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 xVLANTagSetVID()

```
BaseType_t xVLANTagSetVID (
    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.

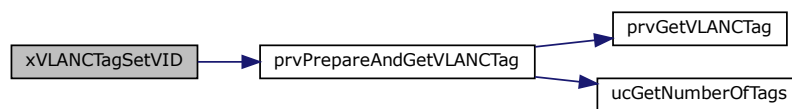
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xClass</i> | 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.

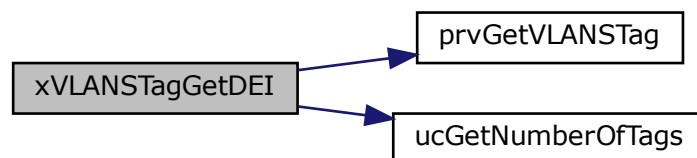
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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.

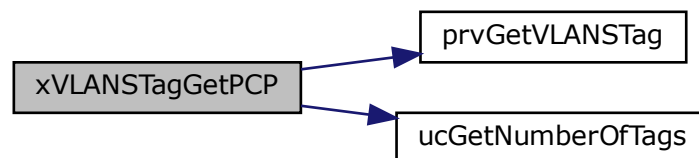
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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.

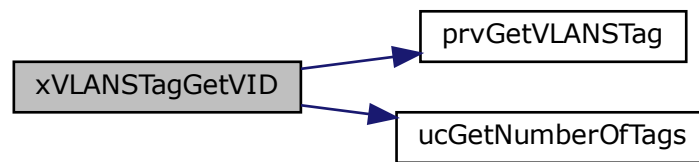
Parameters

| | |
|--------------|---|
| <i>pxBuf</i> | 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

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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.

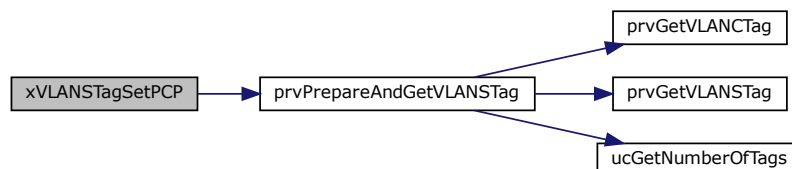
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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.

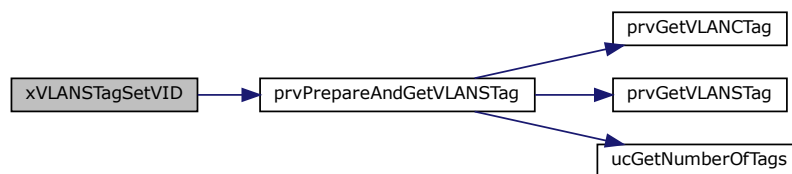
Parameters

| | |
|---------------|---|
| <i>pxBuf</i> | Pointer to the network buffer descriptor. |
| <i>xValue</i> | 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;

```

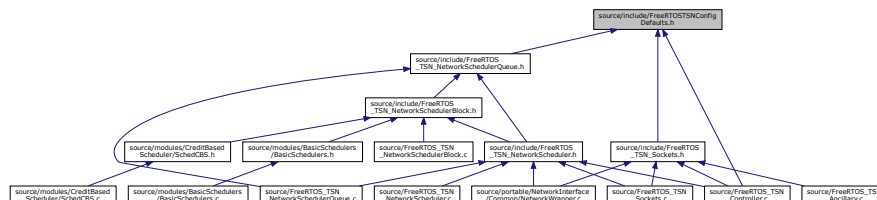
```

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 xVLANSTagGetPCP      xVLANCTagGetPCP
93 #define xVLANSTagGetDEI      xVLANCTagGetDEI
94 #define xVLANSTagGetVID      xVLANCTagGetVID
95 #define xVLANSTagCheckClass  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 xVLANSTagSetPCP      xVLANCTagSetPCP
113 #define xVLANSTagSetDEI      xVLANCTagSetDEI
114 #define xVLANSTagSetVID      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 FreeRTOSConfigDefaults.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 FreeRTOSConfig.h has not been included yet
6 #endif
7
8 #define tsconfigENABLE      ( 1 )
9
10 #define tsconfigDISABLE    ( 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 * tsconfigCONTROLLER_MAX_EVENT_WAIT ticks.
16 */
17 #ifndef tsconfigDEFAULT_QUEUE_TIMEOUT
18 #define tsconfigDEFAULT_QUEUE_TIMEOUT    ( 50U )
19 #endif
20
21 #if ( tsconfigDEFAULT_QUEUE_TIMEOUT < 0 )
22 #error tsconfigDEFAULT_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 tsconfigCONTROLLER_MAX_EVENT_WAIT
31 #define tsconfigCONTROLLER_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 tsconfigMAX_QUEUE_NAME_LEN
38 #define tsconfigMAX_QUEUE_NAME_LEN    ( 32U )
39 #endif
40
41 #if ( tsconfigMAX_QUEUE_NAME_LEN < 0 )
42 #error tsconfigMAX_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 tsconfigINCLUDE_QUEUE_EVENT_CALLBACKS
49 #define tsconfigINCLUDE_QUEUE_EVENT_CALLBACKS    tsconfigDISABLE
50 #endif
51
52 #if ( ( tsconfigINCLUDE_QUEUE_EVENT_CALLBACKS != tsconfigDISABLE ) && (
53     tsconfigINCLUDE_QUEUE_EVENT_CALLBACKS != tsconfigENABLE ) )
54 #error Invalid tsconfigINCLUDE_QUEUE_EVENT_CALLBACKS configuration
55 #endif
56
57 /* Used the IPV field in the network queue structure to implement a priority
58 * inheritance mechanism on the network queues. Note that this is a simplified
59 * version that assumes that whenever a packet is waiting in a queue, a task of
60 * the corresponding priority is always waiting for it. The TSN controller will
61 * then assume a priority which is the maximum IPV of all the queues with
62 * pending messages.
63 */
64 #ifndef tsconfigCONTROLLER_HAS_DYNAMIC_PRIO
65 #define tsconfigCONTROLLER_HAS_DYNAMIC_PRIO    tsconfigDISABLE
66 #endif
67
68 #if ( ( tsconfigCONTROLLER_HAS_DYNAMIC_PRIO != tsconfigDISABLE ) && (
69     tsconfigCONTROLLER_HAS_DYNAMIC_PRIO != tsconfigENABLE ) )
70 #error Invalid tsconfigCONTROLLER_HAS_DYNAMIC_PRIO configuration
71 #endif
72
73 /* FreeRTOS priority of the TSN controller task. If tsconfigCONTROLLER_HAS_DYNAMIC_PRIO
74 * this config entry is ignored as the base priority of the TSN controller is
75 * the priority of the idle task plus 1
76 */
77 #ifndef tsconfigTSN_CONTROLLER_PRIORITY
78 #define tsconfigTSN_CONTROLLER_PRIORITY    ( configMAX_PRIORITIES - 1 )
79 #endif
80
81 #if ( ( tsconfigTSN_CONTROLLER_PRIORITY < 0 ) )

```

```

81 #error Invalid tsconfigTSN_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 tsconfigWRAPPER_INSERTS_VLAN_TAGS
90 #define tsconfigWRAPPER_INSERTS_VLAN_TAGS    tsconfigENABLE
91 #endif
92
93 #if ( ( tsconfigWRAPPER_INSERTS_VLAN_TAGS != tsconfigDISABLE ) && ( tsconfigWRAPPER_INSERTS_VLAN_TAGS
94     != tsconfigENABLE ) )
95 #error Invalid tsconfigWRAPPER_INSERTS_VLAN_TAGS configuration
96 #endif
97
98 /* This option allows the user to set a flag in the sockets to specify the VLAN
99 * tag. This option reduces compatibility with current +TCP sockets and makes
100 * the API different from Linux sockets. This is going to be removed in future,
101 * please consider using tsconfigWRAPPER_INSERTS_VLAN_TAGS instead.
102 */
103 #ifndef tsconfigSOCKET_INSERTS_VLAN_TAGS
104 #define tsconfigSOCKET_INSERTS_VLAN_TAGS    tsconfigDISABLE
105 #endif
106
107 #if ( ( tsconfigSOCKET_INSERTS_VLAN_TAGS != tsconfigDISABLE ) && ( tsconfigSOCKET_INSERTS_VLAN_TAGS
108     != tsconfigENABLE ) )
109 #error Invalid tsconfigSOCKET_INSERTS_VLAN_TAGS configuration
110 #endif
111
112 #if ( ( tsconfigWRAPPER_INSERTS_VLAN_TAGS == tsconfigENABLE ) && ( tsconfigSOCKET_INSERTS_VLAN_TAGS
113     == tsconfigENABLE ) )
114 #error tsconfigWRAPPER_INSERTS_VLAN_TAGS and tsconfigSOCKET_INSERTS_VLAN_TAGS cannot be enabled at the
115     same time
116 #endif
117
118 /* The maximum number of messages waiting in a socket errqueue
119 */
120 #ifndef tsconfigERRQUEUE_LENGTH
121 #define tsconfigERRQUEUE_LENGTH    ( 16 )
122 #endif
123
124 #if ( tsconfigERRQUEUE_LENGTH <= 0 )
125 #error Invalid tsconfigERRQUEUE_LENGTH configuration
126 #endif
127
128 /* Print a dump of ingress/egress packets in hex
129 */
130 #ifndef tsconfigDUMP_PACKETS
131 #define tsconfigDUMP_PACKETS    tsconfigDISABLE
132 #endif
133
134 #if ( ( tsconfigDUMP_PACKETS != tsconfigDISABLE ) && ( tsconfigDUMP_PACKETS != tsconfigENABLE ) )
135 #error Invalid tsconfigDUMP_PACKETS configuration
136 #endif
137
138 #endif /* FREERTOS_TSN_CONFIG_DEFAULTS_H */

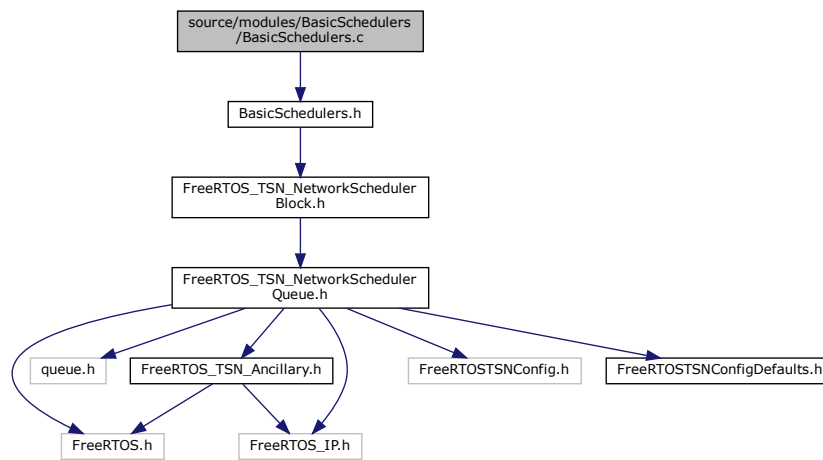
```

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

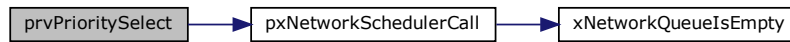
Implementation of some basic schedulers.

5.34.2 Function Documentation

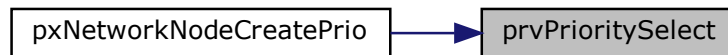
5.34.2.1 prvPrioritySelect()

```
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 )
```

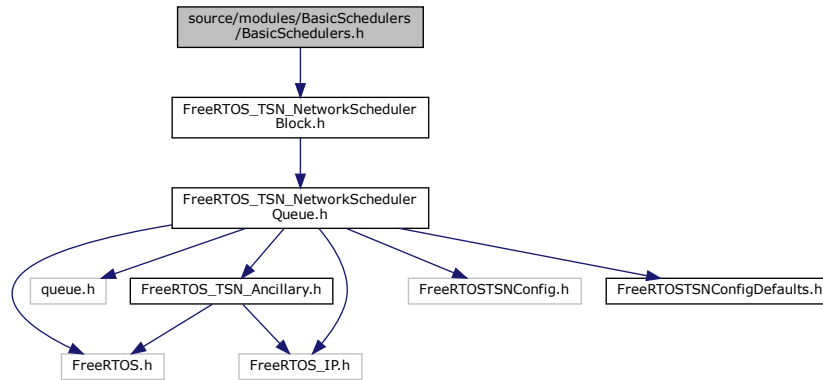
Here is the call graph for this function:



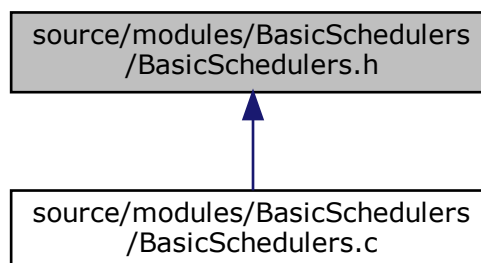
5.35 source/modules/BasicSchedulers/BasicSchedulers.h File Reference

```
#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.](#)

```

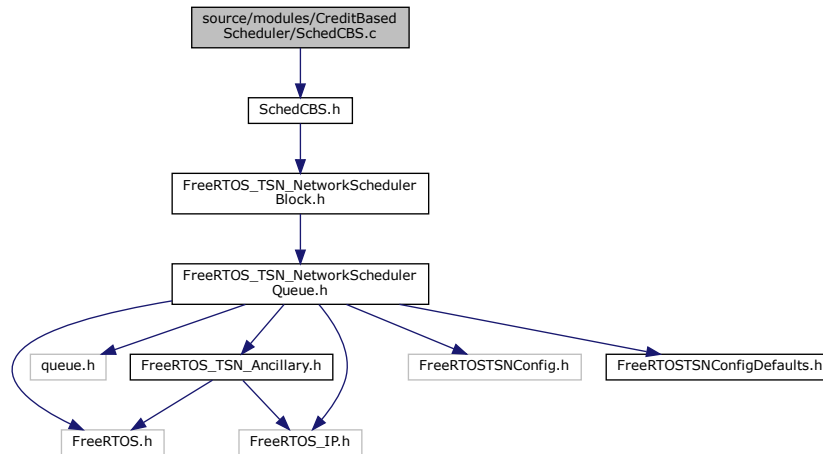
1 #ifndef BASIC_SCHEDULERS_H
2 #define BASIC_SCHEDULERS_H
3
4 #include "FreeRTOS_TSN_NetworkSchedulerBlock.h"
5
6 NetworkNode_t * pxNetworkNodeCreateFIFO( void );
7
8 NetworkNode_t * pxNetworkNodeCreateRR( BaseType_t uxNumChildren );
9
10 NetworkNode_t * pxNetworkNodeCreatePrio( BaseType_t uxNumChildren );
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

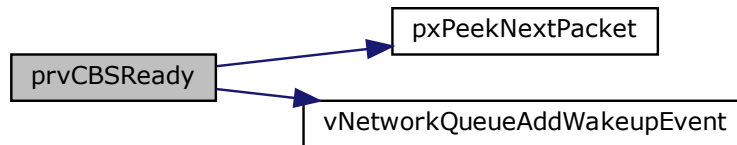
Implementation of a Credit Based Scheduler.

5.37.2 Function Documentation

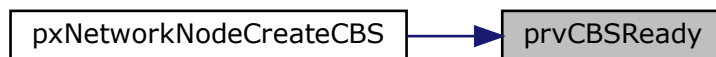
5.37.2.1 prvCBSReady()

```
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.

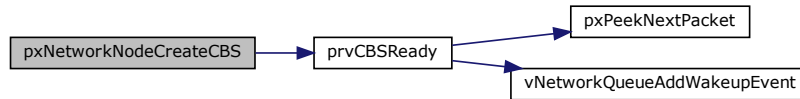
Parameters

| | |
|--------------------|--|
| <i>uxBandwidth</i> | The desired bandwidth of the scheduler, measured in bit per second |
| <i>uxMaxCredit</i> | The max credit of the scheduler, in bits |

Returns

A pointer to the node with the scheduler

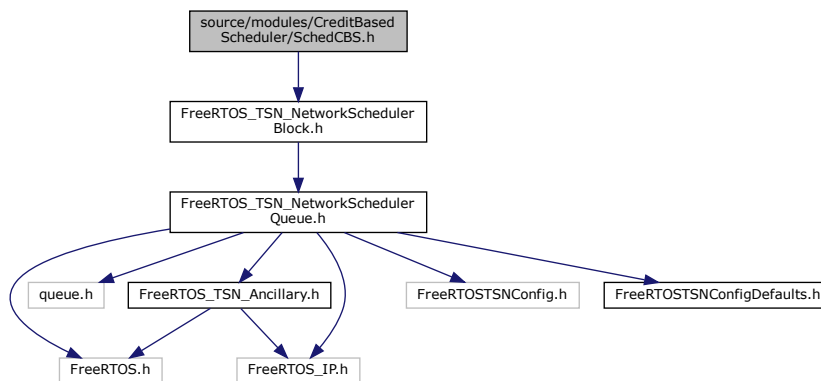
Here is the call graph for this function:



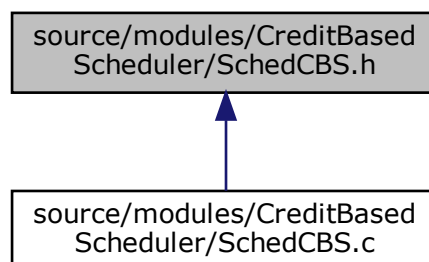
5.38 source/modules/CreditBasedScheduler/SchedCBS.h File Reference

```
#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.

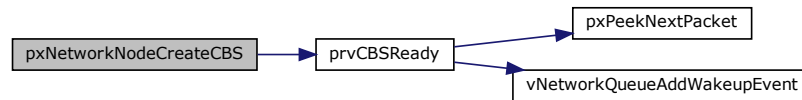
Parameters

| | |
|--------------------|--|
| <i>uxBandwidth</i> | The desired bandwidth of the scheduler, measured in bit per second |
| <i>uxMaxCredit</i> | 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 */

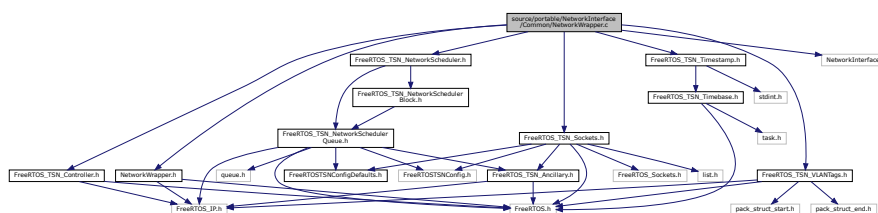
```

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 *pusVLANSVC, uint16_t *pusVLANServiceTCI)
- void [prvDumpPacket](#) (char *const pcPrefix, NetworkBufferDescriptor_t *pxBuf)
- BaseType_t [prvAncillaryMsgControlFillForRx](#) (struct msghdr *pxMsg, NetworkBufferDescriptor_t *pxBuf, Socket_t xSocket, TSNSocket_t xTSNSocket)
- BaseType_t [prvAncillaryMsgControlFillForTx](#) (struct msghdr *pxMsg, 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 *pxBuf, 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 *pxBuf, 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

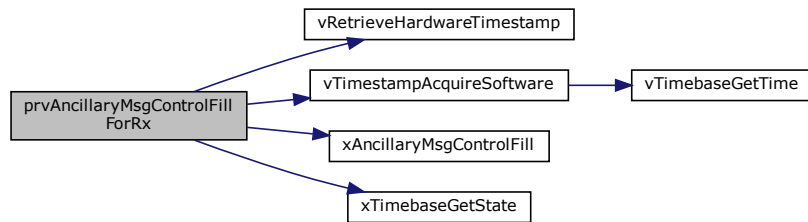
```
#define xSendEventStructToIPTask xSendEventStructToTSNController
```

5.40.3 Function Documentation

5.40.3.1 prvAncillaryMsgControlFillForRx()

```
BaseType_t prvAncillaryMsgControlFillForRx (
    struct msghdr * pxMsg,
    NetworkBufferDescriptor_t * pxBuf,
    Socket_t xSocket,
    TSNSocket_t xTSNSocket )
```

Here is the call graph for this function:



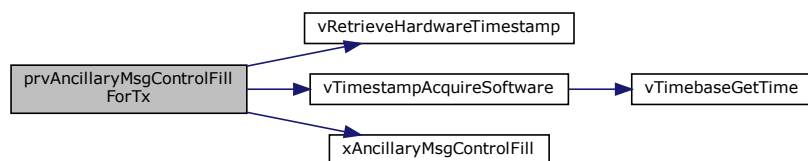
Here is the caller graph for this function:



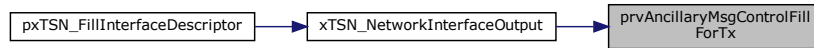
5.40.3.2 prvAncillaryMsgControlFillForTx()

```
BaseType_t prvAncillaryMsgControlFillForTx (
    struct msghdr * pxMsg,
    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:

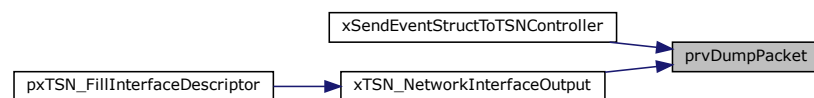


5.40.3.3 prvDumpPacket()

```

void prvDumpPacket (
    char *const pcPrefix,
    NetworkBufferDescriptor_t * pxBuf )
  
```

Here is the caller graph for this function:

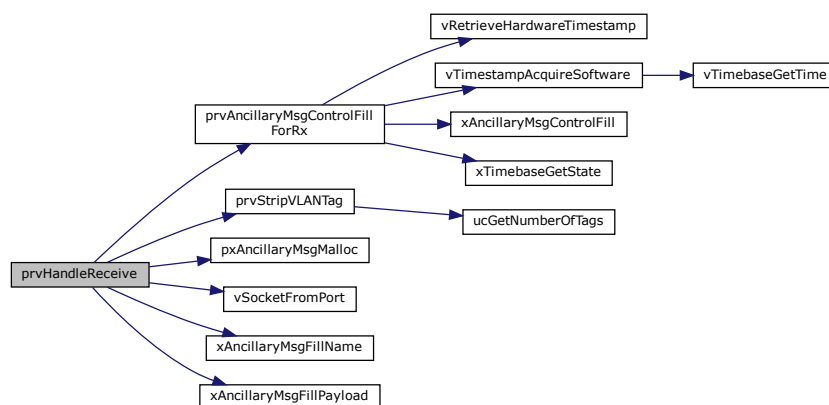


5.40.3.4 prvHandleReceive()

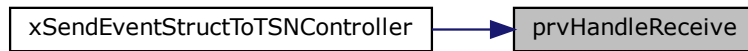
```

NetworkQueueItem_t * prvHandleReceive (
    NetworkBufferDescriptor_t * pxBuf )
  
```

Here is the call graph for this function:



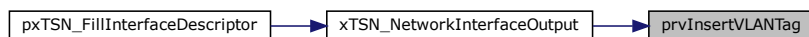
Here is the caller graph for this function:



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 * pusVLANSERVICE )
```

Here is the call graph for this function:

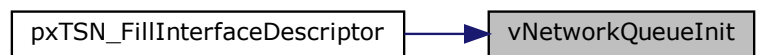


```
graph LR; A[xSendEventStructToTSNController] --> B[prvHandleReceive]; B --> C[prvStripVLANTag];
```

```
NetworkInterface_t * pxTSN_FillInterfaceDescriptor (
    BaseType_t xEMACIndex,
    NetworkInterface_t * pxInterface,
    NetworkInterfaceConfig_t * pxInterfaceConfig )
```

[illegible]

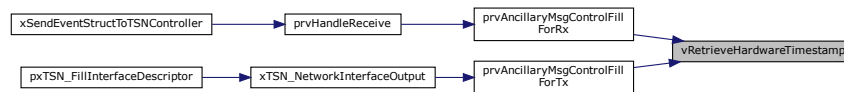
This function should be defined by the user and is project specific Here is the caller graph for this function:



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:

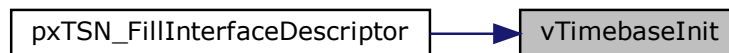


5.40.3.10 vTimebaseInit()

```
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.

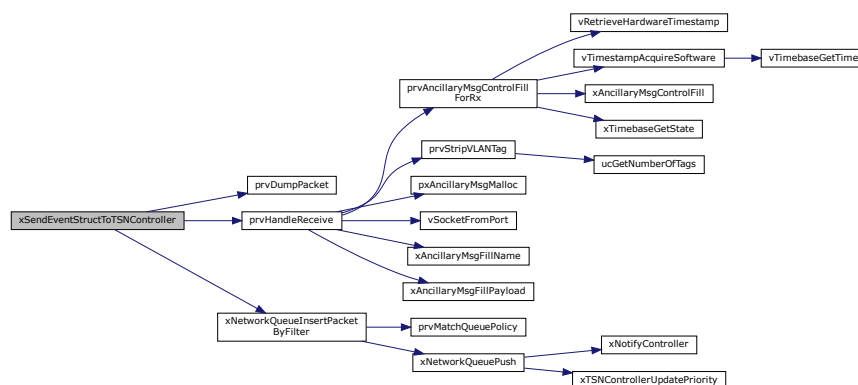
Parameters

| | | |
|----|------------------|---|
| in | <i>pxEvent</i> | Pointer to the IP stack event structure |
| in | <i>uxTimeout</i> | Timeout value for sending the event |

Returns

pdTRUE if the event is sent successfully, pdFALSE otherwise

Here is the call graph for this function:



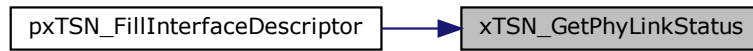
5.40.3.15 xTSN_GetPhyLinkStatus()

```
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:



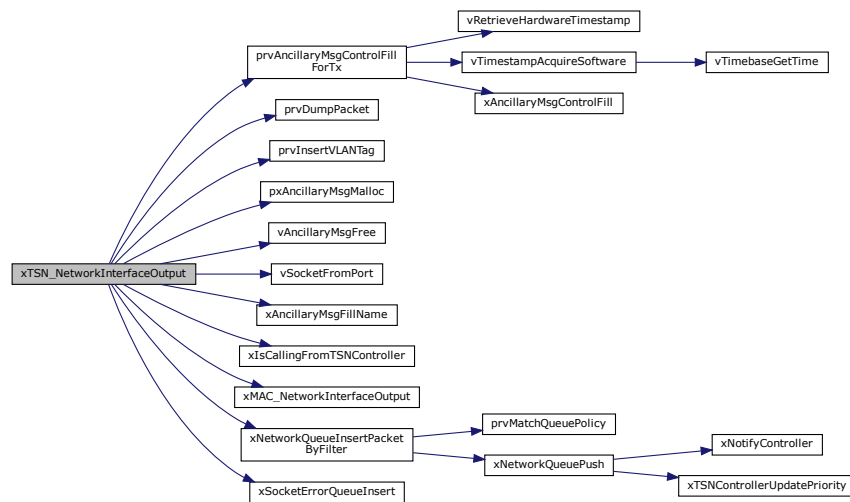
5.40.3.17 xTSN_NetworkInterfaceOutput()

```

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

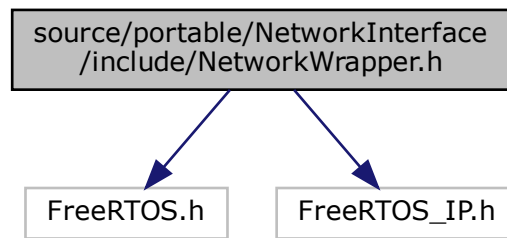
5.40.4.1 xNetworkWrapperInitialised

```
BaseType_t xNetworkWrapperInitialised = pdFALSE
```

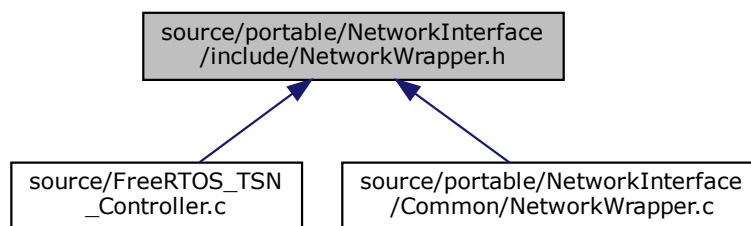
5.41 source/portable/NetworkInterface/include/NetworkWrapper.h File Reference

```
#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](#)

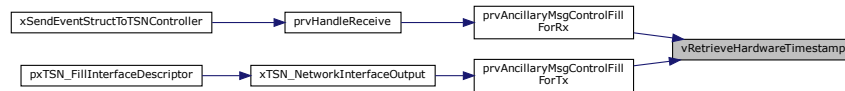
Functions

- BaseType_t [xTSN_NetworkInterfaceInitialise](#) (NetworkInterface_t *pxInterface)
- BaseType_t [xTSN_NetworkInterfaceOutput](#) (NetworkInterface_t *pxInterface, NetworkBufferDescriptor_t *const pBuffer, 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)

5.41.2.2 vRetrieveHardwareTimestamp()

```
void vRetrieveHardwareTimestamp (
    NetworkInterface_t * pxInterface,
    NetworkBufferDescriptor_t * pxBuf,
    uint32_t * pusSec,
    uint32_t * pusNanosec )
```

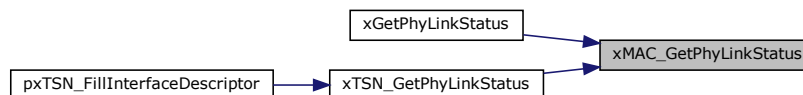
Here is the caller graph for this function:



5.41.2.3 xMAC_GetPhyLinkStatus()

```
BaseType_t xMAC_GetPhyLinkStatus (
    NetworkInterface_t * pxInterface )
```

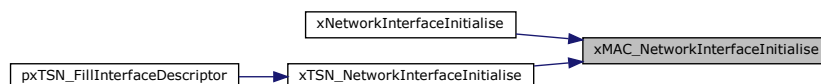
Here is the caller graph for this function:



5.41.2.4 xMAC_NetworkInterfaceInitialise()

```
BaseType_t xMAC_NetworkInterfaceInitialise (
    NetworkInterface_t * pxInterface )
```

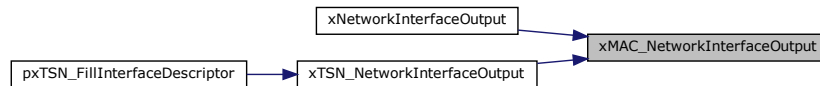
Here is the caller graph for this function:



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.

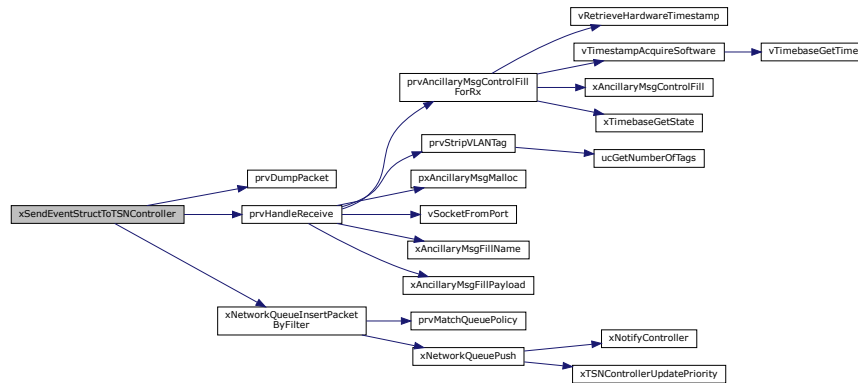
Parameters

| | | |
|----|------------------|---|
| in | <i>pxEvent</i> | Pointer to the IP stack event structure |
| in | <i>uxTimeout</i> | 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()

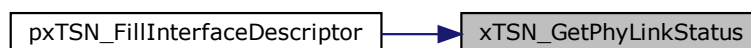
```

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:

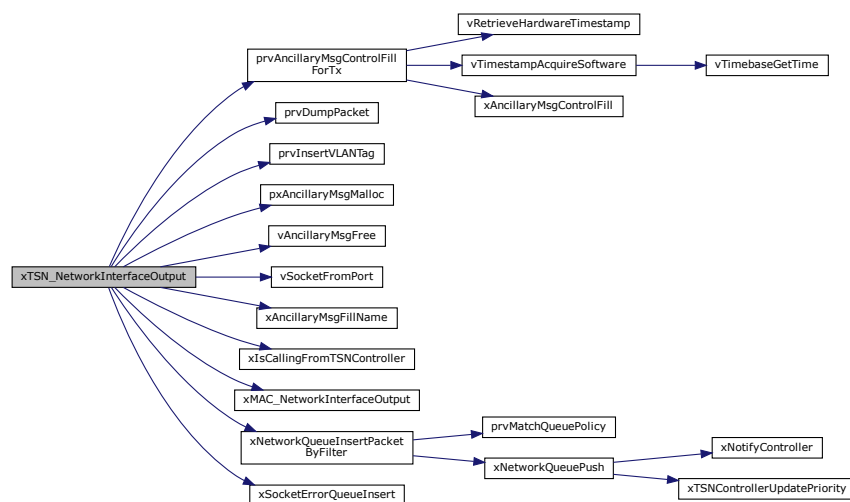


5.41.2.9 xTSN_NetworkInterfaceOutput()

```
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

- __MSG_FIRSTHDR
 - FreeRTOS_TSN_Ancillary.h, [100](#)
 - __MSG_NXTHDR
 - FreeRTOS_TSN_Ancillary.c, [32](#)
 - FreeRTOS_TSN_Ancillary.h, [102](#)
- BasicSchedulers.c
 - prvPrioritySelect, [181](#)
 - pxNetworkNodeCreateFIFO, [182](#)
 - pxNetworkNodeCreatePrio, [182](#)
- BasicSchedulers.h
 - pxNetworkNodeCreateFIFO, [183](#)
 - pxNetworkNodeCreatePrio, [184](#)
 - pxNetworkNodeCreateRR, [184](#)
- MSG_ALIGN
 - FreeRTOS_TSN_Ancillary.h, [100](#)
- MSG_DATA
 - FreeRTOS_TSN_Ancillary.h, [100](#)
- MSG_FIRSTHDR
 - FreeRTOS_TSN_Ancillary.h, [100](#)
- MSG_LEN
 - FreeRTOS_TSN_Ancillary.h, [100](#)
- msg_len
 - msg_hdr, [7](#)
- msg_level
 - msg_hdr, [7](#)
- MSG_NXTHDR
 - FreeRTOS_TSN_Ancillary.h, [101](#)
- MSG_SPACE
 - FreeRTOS_TSN_Ancillary.h, [101](#)
- msg_type
 - msg_hdr, [7](#)
- msg_hdr, [7](#)
 - msg_len, [7](#)
 - msg_level, [7](#)
 - msg_type, [7](#)
- cName
 - xNETQUEUE, [15](#)
- controllerTSN_TASK_BASE_PRIO
 - FreeRTOS_TSN_Controller.c, [39](#)
- diffservCLASS_AFxy
 - FreeRTOS_TSN_DS.h, [114](#)
- diffservCLASS_CSx
 - FreeRTOS_TSN_DS.h, [114](#)
- diffservCLASS_DF
 - FreeRTOS_TSN_DS.h, [114](#)
- diffservCLASS_DSCP_CUSTOM
 - FreeRTOS_TSN_DS.h, [114](#)
- diffservCLASS_EF
 - FreeRTOS_TSN_DS.h, [115](#)
- diffservCLASS_LE
 - FreeRTOS_TSN_DS.h, [115](#)
- diffservGET_DSCLASS_IPv4
 - FreeRTOS_TSN_DS.h, [115](#)
- diffservGET_DSCLASS_IPv6
 - FreeRTOS_TSN_DS.h, [115](#)
- diffservSET_DSCLASS_IPv4
 - FreeRTOS_TSN_DS.h, [115](#)
- diffservSET_DSCLASS_IPv6
 - FreeRTOS_TSN_DS.h, [115](#)
- ee_code
 - sock_extended_err, [12](#)
- ee_data
 - sock_extended_err, [12](#)
- ee_errno
 - sock_extended_err, [12](#)
- ee_info
 - sock_extended_err, [12](#)
- ee_origin
 - sock_extended_err, [12](#)
- ee_pad
 - sock_extended_err, [12](#)
- ee_type
 - sock_extended_err, [13](#)
- eEventType
 - xNETQUEUE_ITEM, [17](#)
- eIPTaskEvents
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, [133](#)
- ePolicy
 - xNETQUEUE, [15](#)
- eQueuePolicy_t
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, [133](#)
- eRecvOnly
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, [133](#)
- eSendOnly
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, [133](#)
- eSendRecv
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, [133](#)
- eTimebaseDisabled
 - FreeRTOS_TSN_Timebase.h, [151](#)
- eTimebaseEnabled
 - FreeRTOS_TSN_Timebase.h, [151](#)
- eTimebaseNotInitialised
 - FreeRTOS_TSN_Timebase.h, [151](#)
- eTimebaseState_t
 - FreeRTOS_TSN_Timebase.h, [151](#)

- FilterFunction_t
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, 133
- fnAdjTime
 - xTIMEBASE, 26
- fnFilter
 - xNETQUEUE, 15
- fnGetTime
 - xTIMEBASE, 26
- fnReady
 - xSCHEDULER_GENERIC, 23
- fnSelect
 - xSCHEDULER_GENERIC, 24
- fnSetTime
 - xTIMEBASE, 27
- fnStart
 - xTIMEBASE, 27
- fnStop
 - xTIMEBASE, 27
- FREERTOS_IP_RECVERR
 - FreeRTOS_TSN_Sockets.h, 137
- FREERTOS_IPV6_RECVERR
 - FreeRTOS_TSN_Sockets.h, 138
- FREERTOS_MSG_ERRQUEUE
 - FreeRTOS_TSN_Sockets.h, 138
- FREERTOS_SCM_TIMESTAMP
 - FreeRTOS_TSN_Sockets.h, 138
- FREERTOS_SCM_TIMESTAMPING
 - FreeRTOS_TSN_Sockets.h, 138
- freertos_scm_timestamping, 8
 - ts, 8
- FREERTOS_SCM_TIMESTAMPNS
 - FreeRTOS_TSN_Sockets.h, 138
- FREERTOS_SO_DS_CLASS
 - FreeRTOS_TSN_Sockets.h, 138
- FREERTOS_SO_TIMESTAMP
 - FreeRTOS_TSN_Sockets.h, 138
- FREERTOS_SO_TIMESTAMP_OLD
 - FreeRTOS_TSN_Sockets.h, 138
- FREERTOS_SO_TIMESTAMPING
 - FreeRTOS_TSN_Sockets.h, 139
- FREERTOS_SO_TIMESTAMPING_OLD
 - FreeRTOS_TSN_Sockets.h, 139
- FREERTOS_SO_TIMESTAMPNS
 - FreeRTOS_TSN_Sockets.h, 139
- FREERTOS_SO_TIMESTAMPNS_OLD
 - FreeRTOS_TSN_Sockets.h, 139
- FREERTOS_SOL_IP
 - FreeRTOS_TSN_Sockets.h, 139
- FREERTOS_SOL_IPV6
 - FreeRTOS_TSN_Sockets.h, 139
- FREERTOS_SOL_SOCKET
 - FreeRTOS_TSN_Sockets.h, 139
- freertos_timespec, 8
 - tv_nsec, 9
 - tv_sec, 9
- FreeRTOS_TSN_Ancillary.c
 - __CMSG_NXTHDR, 32
 - pxAncillaryMsgMalloc, 32
 - vAncillaryMsgFree, 33
 - vAncillaryMsgFreeAll, 33
 - vAncillaryMsgFreeControl, 34
 - vAncillaryMsgFreeName, 34
 - vAncillaryMsgFreePayload, 35
 - xAncillaryMsgControlFill, 35
 - xAncillaryMsgControlFillSingle, 36
 - xAncillaryMsgFillName, 36
 - xAncillaryMsgFillPayload, 37
- FreeRTOS_TSN_Ancillary.h
 - __CMSG_FIRSTHDR, 100
 - __CMSG_NXTHDR, 102
 - CMSG_ALIGN, 100
 - CMSG_DATA, 100
 - CMSG_FIRSTHDR, 100
 - CMSG_LEN, 100
 - CMSG_NXTHDR, 101
 - CMSG_SPACE, 101
 - pdFREERTOS_ERRNO_ENOMSG, 101
 - pxAncillaryMsgMalloc, 102
 - SO_EE_ORIGIN_ICMP, 101
 - SO_EE_ORIGIN_ICMP6, 101
 - SO_EE_ORIGIN_LOCAL, 101
 - SO_EE_ORIGIN_NONE, 101
 - SO_EE_ORIGIN_TIMESTAMPING, 102
 - SO_EE_ORIGIN_TXSTATUS, 102
 - SO_EE_ORIGIN_TXTIME, 102
 - SO_EE_ORIGIN_ZEROCOPY, 102
 - vAncillaryMsgFree, 103
 - vAncillaryMsgFreeAll, 103
 - vAncillaryMsgFreeControl, 104
 - vAncillaryMsgFreeName, 105
 - vAncillaryMsgFreePayload, 105
 - xAncillaryMsgControlFill, 105
 - xAncillaryMsgControlFillSingle, 106
 - xAncillaryMsgFillName, 106
 - xAncillaryMsgFillPayload, 107
- FreeRTOS_TSN_bind
 - FreeRTOS_TSN_Sockets.c, 66
 - FreeRTOS_TSN_Sockets.h, 141
- FreeRTOS_TSN_closesocket
 - FreeRTOS_TSN_Sockets.c, 67
 - FreeRTOS_TSN_Sockets.h, 141
- FreeRTOS_TSN_Controller.c
 - controllerTSN_TASK_BASE_Prio, 39
 - prvDeliverFrame, 39
 - prvReceiveUDPPacketTSN, 40
 - prvTSNController, 41
 - pxNetworkQueueList, 45
 - vTSNController_Initialise, 42
 - vTSNControllerComputePriority, 42
 - xlsCallingFromTSNController, 43
 - xNotifyController, 43
 - xTSNControllerHandle, 45
 - xTSNControllerUpdatePriority, 44
- FreeRTOS_TSN_Controller.h
 - vTSNController_Initialise, 110
 - vTSNControllerComputePriority, 110

- xlsCallingFromTSNController, 111
 - xNotifyController, 111
 - xTSNControllerUpdatePriority, 112
- FreeRTOS_TSN_DS.c
 - prvGetIPVersionAndOffset, 46
 - ucDSCClassGet, 46
 - xDSCClassSet, 47
- FreeRTOS_TSN_DS.h
 - diffservCLASS_AFxy, 114
 - diffservCLASS_CSx, 114
 - diffservCLASS_DF, 114
 - diffservCLASS_DSCP_CUSTOM, 114
 - diffservCLASS_EF, 115
 - diffservCLASS_LE, 115
 - diffservGET_DSCLASS_IPv4, 115
 - diffservGET_DSCLASS_IPv6, 115
 - diffservSET_DSCLASS_IPv4, 115
 - diffservSET_DSCLASS_IPv6, 115
 - ucDSCClassGet, 116
 - xDSCClassSet, 116
- FREERTOS_TSN_INVALID_SOCKET
 - FreeRTOS_TSN_Sockets.h, 139
- FreeRTOS_TSN_NetworkScheduler.c
 - prvMatchQueuePolicy, 49
 - pxNetworkQueueList, 54
 - pxNetworkQueueRoot, 54
 - uxNumQueues, 54
 - vNetworkQueueListAdd, 49
 - xNetworkQueueAssignRoot, 50
 - xNetworkQueueInsertPacketByFilter, 50
 - xNetworkQueueInsertPacketByName, 51
 - xNetworkQueuePop, 52
 - xNetworkQueuePush, 52
 - xNetworkQueueSchedule, 53
- FreeRTOS_TSN_NetworkScheduler.h
 - NetworkQueueList_t, 119
 - pxNetworkQueueFindByName, 119
 - vNetworkQueueInit, 119
 - vNetworkQueueListAdd, 120
 - xNetworkQueueAssignRoot, 120
 - xNetworkQueueInsertPacketByFilter, 120
 - xNetworkQueueInsertPacketByName, 121
 - xNetworkQueuePop, 122
 - xNetworkQueuePush, 122
 - xNetworkQueueSchedule, 123
- FreeRTOS_TSN_NetworkSchedulerBlock.c
 - prvAlwaysReady, 56
 - prvSelectFirst, 56
 - pxNetworkSchedulerCall, 56
 - pxPeekNextPacket, 58
 - uxNetworkQueueGetTicksUntilWakeup, 59
 - uxNextWakeup, 61
 - vNetworkQueueAddWakeupEvent, 59
 - xNetworkSchedulerLinkChild, 60
 - xNetworkSchedulerLinkQueue, 60
- FreeRTOS_TSN_NetworkSchedulerBlock.h
 - netschedCALL_READY_FROM_NODE, 126
 - netschedCALL_SELECT_FROM_NODE, 126
- NetworkNode_t, 127
- pxNetworkSchedulerCall, 127
- pxPeekNextPacket, 128
- ReadyQueueFunction_t, 127
- SelectQueueFunction_t, 127
- uxNetworkQueueGetTicksUntilWakeup, 129
- vNetworkQueueAddWakeupEvent, 129
- xNetworkSchedulerLinkChild, 129
- xNetworkSchedulerLinkQueue, 130
- FreeRTOS_TSN_NetworkSchedulerQueue.c
 - prvAlwaysTrue, 62
 - prvDefaultPacketHandler, 62
 - uxNetworkQueuePacketsWaiting, 63
 - xNetworkQueueIsEmpty, 63
- FreeRTOS_TSN_NetworkSchedulerQueue.h
 - eIPTaskEvents, 133
 - eQueuePolicy_t, 133
 - eRecvOnly, 133
 - eSendOnly, 133
 - eSendRecv, 133
 - FilterFunction_t, 133
 - NetworkQueue_t, 133
 - NetworkQueueItem_t, 133
 - PacketHandleFunction_t, 133
 - uxNetworkQueuePacketsWaiting, 134
 - xNetworkQueueIsEmpty, 134
- FreeRTOS_TSN_recvfrom
 - FreeRTOS_TSN_Sockets.c, 67
 - FreeRTOS_TSN_Sockets.h, 142
- FreeRTOS_TSN_recvmsg
 - FreeRTOS_TSN_Sockets.c, 68
 - FreeRTOS_TSN_Sockets.h, 142
- FreeRTOS_TSN_sendto
 - FreeRTOS_TSN_Sockets.c, 69
 - FreeRTOS_TSN_Sockets.h, 143
- FreeRTOS_TSN_setsockopt
 - FreeRTOS_TSN_Sockets.c, 69
 - FreeRTOS_TSN_Sockets.h, 144
- FreeRTOS_TSN_socket
 - FreeRTOS_TSN_Sockets.c, 70
 - FreeRTOS_TSN_Sockets.h, 145
- FreeRTOS_TSN_Socket_t
 - FreeRTOS_TSN_Sockets.h, 140
- FreeRTOS_TSN_Sockets.c
 - FreeRTOS_TSN_bind, 66
 - FreeRTOS_TSN_closesocket, 67
 - FreeRTOS_TSN_recvfrom, 67
 - FreeRTOS_TSN_recvmsg, 68
 - FreeRTOS_TSN_sendto, 69
 - FreeRTOS_TSN_setsockopt, 69
 - FreeRTOS_TSN_socket, 70
 - prvMoveToStartOfPayload, 70
 - prvPrepareBufferUDPv4, 71
 - prvPrepareBufferUDPv6, 72
 - tsnsocketGET_SOCKET_PORT, 65
 - tsnsocketSET_SOCKET_PORT, 66
 - tsnsocketSOCKET_IS_BOUND, 66
 - vInitialiseTSNSockets, 73

- vSocketFromPort, 73
- xSocketErrorQueueInsert, 73
- xTSNBoundUDPSocketList, 74
- FreeRTOS_TSN_Sockets.h
 - FREERTOS_IP_RECVERR, 137
 - FREERTOS_IPV6_RECVERR, 138
 - FREERTOS_MSG_ERRQUEUE, 138
 - FREERTOS_SCM_TIMESTAMP, 138
 - FREERTOS_SCM_TIMESTAMPING, 138
 - FREERTOS_SCM_TIMESTAMPNS, 138
 - FREERTOS_SO_DS_CLASS, 138
 - FREERTOS_SO_TIMESTAMP, 138
 - FREERTOS_SO_TIMESTAMP_OLD, 138
 - FREERTOS_SO_TIMESTAMPING, 139
 - FREERTOS_SO_TIMESTAMPING_OLD, 139
 - FREERTOS_SO_TIMESTAMPNS, 139
 - FREERTOS_SO_TIMESTAMPNS_OLD, 139
 - FREERTOS_SOL_IP, 139
 - FREERTOS_SOL_IPV6, 139
 - FREERTOS_SOL_SOCKET, 139
 - FreeRTOS_TSN_bind, 141
 - FreeRTOS_TSN_closesocket, 141
 - FREERTOS_TSN_INVALID_SOCKET, 139
 - FreeRTOS_TSN_recvfrom, 142
 - FreeRTOS_TSN_recvmsg, 142
 - FreeRTOS_TSN_sendto, 143
 - FreeRTOS_TSN_setsockopt, 144
 - FreeRTOS_TSN_socket, 145
 - FreeRTOS_TSN_Socket_t, 140
 - SCM_TSTAMP_ACK, 141
 - SCM_TSTAMP_SCHED, 141
 - SCM_TSTAMP_SND, 141
 - SOF_TIMESTAMPING_BIND_PHC, 140
 - SOF_TIMESTAMPING_LAST, 140
 - SOF_TIMESTAMPING_MASK, 140
 - SOF_TIMESTAMPING_OPT_CMSG, 140
 - SOF_TIMESTAMPING_OPT_ID, 140
 - SOF_TIMESTAMPING_OPT_ID_TCP, 140
 - SOF_TIMESTAMPING_OPT_PKTINFO, 140
 - SOF_TIMESTAMPING_OPT_STATS, 140
 - SOF_TIMESTAMPING_OPT_TSONLY, 140
 - SOF_TIMESTAMPING_OPT_TX_SWHW, 140
 - SOF_TIMESTAMPING_RAW_HARDWARE, 140
 - SOF_TIMESTAMPING_RX_HARDWARE, 140
 - SOF_TIMESTAMPING_RX_SOFTWARE, 140
 - SOF_TIMESTAMPING_SOFTWARE, 140
 - SOF_TIMESTAMPING_SYS_HARDWARE, 140
 - SOF_TIMESTAMPING_TX_ACK, 140
 - SOF_TIMESTAMPING_TX_HARDWARE, 140
 - SOF_TIMESTAMPING_TX_SCHED, 140
 - SOF_TIMESTAMPING_TX_SOFTWARE, 140
 - TSNSocket_t, 140
 - vInitialiseTSNSockets, 145
 - vSocketFromPort, 146
 - xSocketErrorQueueInsert, 146
- FreeRTOS_TSN_Timebase.c
 - NS_IN_ONE_SEC, 75
 - vTimebaseAdjTime, 76
 - vTimebaseGetTime, 76
 - vTimebaseSetTime, 76
 - vTimebaseStart, 76
 - xTimebaseGetState, 77
 - xTimebaseHandle, 80
 - xTimebaseHandleSet, 77
 - xTimebaseState, 80
 - xTimespecCmp, 77
 - xTimespecDiff, 78
 - xTimespecDiv, 78
 - xTimespecSum, 79
- FreeRTOS_TSN_Timebase.h
 - eTimebaseDisabled, 151
 - eTimebaseEnabled, 151
 - eTimebaseNotInitialised, 151
 - eTimebaseState_t, 151
 - TimeBaseAdjTimeFunction_t, 150
 - TimeBaseGetTimeFunction_t, 150
 - TimebaseHandle_t, 150
 - TimeBaseSetTimeFunction_t, 150
 - TimeBaseStartFunction_t, 151
 - TimeBaseStopFunction_t, 151
 - vTimebaseAdjTime, 151
 - vTimebaseGetTime, 151
 - vTimebaseInit, 153
 - vTimebaseSetTime, 153
 - vTimebaseStart, 153
 - vTimebaseStop, 153
 - xTimebaseGetState, 154
 - xTimebaseHandleSet, 154
 - xTimespecCmp, 154
 - xTimespecDiff, 155
 - xTimespecDiv, 155
 - xTimespecSum, 156
- FreeRTOS_TSN_Timestamp.c
 - vTimestampAcquireSoftware, 81
- FreeRTOS_TSN_Timestamp.h
 - vTimestampAcquireSoftware, 158
- FreeRTOS_TSN_VLANTags.c
 - prvGetVLANCTag, 83
 - prvGetVLANSTag, 84
 - prvPrepareAndGetVLANCTag, 85
 - prvPrepareAndGetVLANSTag, 86
 - ucGetNumberOfTags, 87
 - xVLANCTagCheckClass, 88
 - xVLANCTagGetDEI, 89
 - xVLANCTagGetPCP, 89
 - xVLANCTagGetVID, 90
 - xVLANCTagSetDEI, 91
 - xVLANCTagSetPCP, 92
 - xVLANCTagSetVID, 92
 - xVLANSTagCheckClass, 93
 - xVLANSTagGetDEI, 94
 - xVLANSTagGetPCP, 94
 - xVLANSTagGetVID, 95
 - xVLANSTagSetDEI, 96
 - xVLANSTagSetPCP, 97
 - xVLANSTagSetVID, 97

- FreeRTOS_TSN_VLANTags.h
 - TaggedEthernetHeader_t, [166](#)
 - ucGetNumberOfTags, [166](#)
 - usFrameType, [174](#)
 - vlantagCLASS_0, [162](#)
 - vlantagCLASS_1, [162](#)
 - vlantagCLASS_2, [163](#)
 - vlantagCLASS_3, [163](#)
 - vlantagCLASS_4, [163](#)
 - vlantagCLASS_5, [163](#)
 - vlantagCLASS_6, [163](#)
 - vlantagCLASS_7, [163](#)
 - vlantagDEI_BIT_MASK, [163](#)
 - vlantagETH_TAG_OFFSET, [163](#)
 - vlantagGET_DEI_FROM_TCI, [164](#)
 - vlantagGET_PCP_FROM_TCI, [164](#)
 - vlantagGET_VID_FROM_TCI, [164](#)
 - vlantagPCP_BIT_MASK, [164](#)
 - vlantagSET_DEI_FROM_TCI, [164](#)
 - vlantagSET_PCP_FROM_TCI, [164](#)
 - vlantagSET_VID_FROM_TCI, [165](#)
 - vlantagTPID_DEFAULT, [165](#)
 - vlantagTPID_DOUBLE_TAG, [165](#)
 - vlantagVID_BIT_MASK, [165](#)
 - xDestinationAddress, [174](#)
 - xSourceAddress, [174](#)
 - xVLANCtagSetDEI, [167](#)
 - xVLANCtagSetPCP, [168](#)
 - xVLANCtagSetVID, [168](#)
 - xVLANStagCheckClass, [169](#)
 - xVLANStagGetDEI, [170](#)
 - xVLANStagGetPCP, [170](#)
 - xVLANStagGetVID, [171](#)
 - xVLANStagSetDEI, [172](#)
 - xVLANStagSetPCP, [173](#)
 - xVLANStagSetVID, [173](#)
 - xVLANTag, [175](#)
 - xVLANTagCheckClass, [165](#)
 - xVLANTagGetDEI, [165](#)
 - xVLANTagGetPCP, [166](#)
 - xVLANTagGetVID, [166](#)
 - xVLANTagSetDEI, [166](#)
 - xVLANTagSetPCP, [166](#)
 - xVLANTagSetVID, [166](#)
- FreeRTOS_TSN_ConfigDefaults.h
 - tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO, [177](#)
 - tsnconfigCONTROLLER_MAX_EVENT_WAIT, [177](#)
 - tsnconfigDEFAULT_QUEUE_TIMEOUT, [177](#)
 - tsnconfigDISABLE, [177](#)
 - tsnconfigDUMP_PACKETS, [177](#)
 - tsnconfigENABLE, [177](#)
 - tsnconfigERRQUEUE_LENGTH, [178](#)
 - tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS, [178](#)
 - tsnconfigMAX_QUEUE_NAME_LEN, [178](#)
 - tsnconfigSOCKET_INSERTS_VLAN_TAGS, [178](#)
 - tsnconfigTSN_CONTROLLER_PRIORITY, [178](#)
 - tsnconfigWRAPPER_INSERTS_VLAN_TAGS, [178](#)
- iov_base
 - iovec, [9](#)
- iov_len
 - iovec, [9](#)
- iovec, [9](#)
 - iov_base, [9](#)
 - iov_len, [9](#)
- msg_control
 - msg_hdr, [10](#)
- msg_controllen
 - msg_hdr, [10](#)
- msg_flags
 - msg_hdr, [11](#)
- msg_iov
 - msg_hdr, [11](#)
- msg_iovlen
 - msg_hdr, [11](#)
- msg_name
 - msg_hdr, [11](#)
- msg_namelen
 - msg_hdr, [11](#)
- msg_hdr, [10](#)
 - msg_control, [10](#)
 - msg_controllen, [10](#)
 - msg_flags, [11](#)
 - msg_iov, [11](#)
 - msg_iovlen, [11](#)
 - msg_name, [11](#)
 - msg_namelen, [11](#)
- netschedCALL_READY_FROM_NODE
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, [126](#)
- netschedCALL_SELECT_FROM_NODE
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, [126](#)
- netschedCBS_DEFAULT_BANDWIDTH
 - SchedCBS.h, [188](#)
- netschedCBS_DEFAULT_MAXCREDIT
 - SchedCBS.h, [188](#)
- NetworkInterfaceConfig_t
 - NetworkWrapper.h, [202](#)
- NetworkNode_t
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, [127](#)
- NetworkQueue_t
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, [133](#)
- NetworkQueueItem_t
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, [133](#)
- NetworkQueueList_t
 - FreeRTOS_TSN_NetworkScheduler.h, [119](#)
- NetworkWrapper.c
 - prvAncillaryMsgControlFillForRx, [191](#)
 - prvAncillaryMsgControlFillForTx, [192](#)
 - prvDumpPacket, [193](#)
 - prvHandleReceive, [193](#)
 - prvInsertVLANTag, [194](#)
 - prvStripVLANTag, [194](#)

- pxFillInterfaceDescriptor, 190
- pxTSN_FillInterfaceDescriptor, 195
- vNetworkQueueInit, 195
- vRetrieveHardwareTimestamp, 196
- vTimebaseInit, 196
- wrapperFIRST_TPID, 190
- wrapperSECOND_TPID, 191
- xGetPhyLinkStatus, 191, 196
- xNetworkInterfaceInitialise, 191, 197
- xNetworkInterfaceOutput, 191, 197
- xNetworkWrapperInitialised, 200
- xSendEventStructToIPTask, 191
- xSendEventStructToTSNController, 197
- xTSN_GetPhyLinkStatus, 198
- xTSN_NetworkInterfaceInitialise, 199
- xTSN_NetworkInterfaceOutput, 199
- NetworkWrapper.h
 - NetworkInterfaceConfig_t, 202
 - pxTSN_FillInterfaceDescriptor, 202
 - vRetrieveHardwareTimestamp, 202
 - xMAC_GetPhyLinkStatus, 203
 - xMAC_NetworkInterfaceInitialise, 203
 - xMAC_NetworkInterfaceOutput, 203
 - xSendEventStructToTSNController, 204
 - xTSN_GetPhyLinkStatus, 205
 - xTSN_NetworkInterfaceInitialise, 205
 - xTSN_NetworkInterfaceOutput, 206
- NS_IN_ONE_SEC
 - FreeRTOS_TSN_Timebase.c, 75
- PacketHandleFunction_t
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, 133
- pdFREERTOS_ERRNO_ENOMSG
 - FreeRTOS_TSN_Ancillary.h, 101
- prvAlwaysReady
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, 56
- prvAlwaysTrue
 - FreeRTOS_TSN_NetworkSchedulerQueue.c, 62
- prvAncillaryMsgControlFillForRx
 - NetworkWrapper.c, 191
- prvAncillaryMsgControlFillForTx
 - NetworkWrapper.c, 192
- prvCBSReady
 - SchedCBS.c, 185
- prvDefaultPacketHandler
 - FreeRTOS_TSN_NetworkSchedulerQueue.c, 62
- prvDeliverFrame
 - FreeRTOS_TSN_Controller.c, 39
- prvDumpPacket
 - NetworkWrapper.c, 193
- prvGetIPVersionAndOffset
 - FreeRTOS_TSN_DS.c, 46
- prvGetVLANCTag
 - FreeRTOS_TSN_VLANTags.c, 83
- prvGetVLANSTag
 - FreeRTOS_TSN_VLANTags.c, 84
- prvHandleReceive
 - NetworkWrapper.c, 193
- prvInsertVLANTag
 - NetworkWrapper.c, 194
- prvMatchQueuePolicy
 - FreeRTOS_TSN_NetworkScheduler.c, 49
- prvMoveToStartOfPayload
 - FreeRTOS_TSN_Sockets.c, 70
- prvPrepareAndGetVLANCTag
 - FreeRTOS_TSN_VLANTags.c, 85
- prvPrepareAndGetVLANSTag
 - FreeRTOS_TSN_VLANTags.c, 86
- prvPrepareBufferUDIPv4
 - FreeRTOS_TSN_Sockets.c, 71
- prvPrepareBufferUDIPv6
 - FreeRTOS_TSN_Sockets.c, 72
- prvPrioritySelect
 - BasicSchedulers.c, 181
- prvReceiveUDPPacketTSN
 - FreeRTOS_TSN_Controller.c, 40
- prvSelectFirst
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, 56
- prvStripVLANTag
 - NetworkWrapper.c, 194
- prvTSNController
 - FreeRTOS_TSN_Controller.c, 41
- pvScheduler
 - xNETQUEUE_NODE, 18
- pxAncillaryMsgMalloc
 - FreeRTOS_TSN_Ancillary.c, 32
 - FreeRTOS_TSN_Ancillary.h, 102
- pxBuf
 - xNETQUEUE_ITEM, 17
- pxFillInterfaceDescriptor
 - NetworkWrapper.c, 190
- pxMsg
 - xNETQUEUE_ITEM, 17
- pxNetworkNodeCreateCBS
 - SchedCBS.c, 186
 - SchedCBS.h, 188
- pxNetworkNodeCreateFIFO
 - BasicSchedulers.c, 182
 - BasicSchedulers.h, 183
- pxNetworkNodeCreatePrio
 - BasicSchedulers.c, 182
 - BasicSchedulers.h, 184
- pxNetworkNodeCreateRR
 - BasicSchedulers.h, 184
- pxNetworkQueueFindByName
 - FreeRTOS_TSN_NetworkScheduler.h, 119
- pxNetworkQueueList
 - FreeRTOS_TSN_Controller.c, 45
 - FreeRTOS_TSN_NetworkScheduler.c, 54
- pxNetworkQueueRoot
 - FreeRTOS_TSN_NetworkScheduler.c, 54
- pxNetworkSchedulerCall
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, 56
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, 127
- pxNext
 - xNETQUEUE_NODE, 18
 - xQUEUE_LIST, 20

- pxOwner
 - xSCHEDULER_GENERIC, [24](#)
- pxPeekNextPacket
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, [58](#)
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, [128](#)
- pxQueue
 - xNETQUEUE_NODE, [18](#)
 - xQUEUE_LIST, [20](#)
- pxTSN_FillInterfaceDescriptor
 - NetworkWrapper.c, [195](#)
 - NetworkWrapper.h, [202](#)
- README.md, [31](#)
- ReadyQueueFunction_t
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, [127](#)
- SchedCBS.c
 - prvCBSReady, [185](#)
 - pxNetworkNodeCreateCBS, [186](#)
- SchedCBS.h
 - netschedCBS_DEFAULT_BANDWIDTH, [188](#)
 - netschedCBS_DEFAULT_MAXCREDIT, [188](#)
 - pxNetworkNodeCreateCBS, [188](#)
- SCM_TSTAMP_ACK
 - FreeRTOS_TSN_Sockets.h, [141](#)
- SCM_TSTAMP_SCHED
 - FreeRTOS_TSN_Sockets.h, [141](#)
- SCM_TSTAMP_SND
 - FreeRTOS_TSN_Sockets.h, [141](#)
- SelectQueueFunction_t
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, [127](#)
- SO_EE_ORIGIN_ICMP
 - FreeRTOS_TSN_Ancillary.h, [101](#)
- SO_EE_ORIGIN_ICMP6
 - FreeRTOS_TSN_Ancillary.h, [101](#)
- SO_EE_ORIGIN_LOCAL
 - FreeRTOS_TSN_Ancillary.h, [101](#)
- SO_EE_ORIGIN_NONE
 - FreeRTOS_TSN_Ancillary.h, [101](#)
- SO_EE_ORIGIN_TIMESTAMPING
 - FreeRTOS_TSN_Ancillary.h, [102](#)
- SO_EE_ORIGIN_TXSTATUS
 - FreeRTOS_TSN_Ancillary.h, [102](#)
- SO_EE_ORIGIN_TXTIME
 - FreeRTOS_TSN_Ancillary.h, [102](#)
- SO_EE_ORIGIN_ZEROCOPY
 - FreeRTOS_TSN_Ancillary.h, [102](#)
- sock_extended_err, [11](#)
 - ee_code, [12](#)
 - ee_data, [12](#)
 - ee_errno, [12](#)
 - ee_info, [12](#)
 - ee_origin, [12](#)
 - ee_pad, [12](#)
 - ee_type, [13](#)
- SOF_TIMESTAMPING_BIND_PHC
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_LAST
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_MASK
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_OPT_CMSG
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_OPT_ID
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_OPT_ID_TCP
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_OPT_PKTINFO
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_OPT_STATS
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_OPT_TSONLY
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_OPT_TX_SWHW
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_RAW_HARDWARE
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_RX_HARDWARE
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_RX_SOFTWARE
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_SOFTWARE
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_SYS_HARDWARE
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_TX_ACK
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_TX_HARDWARE
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_TX_SCHED
 - FreeRTOS_TSN_Sockets.h, [140](#)
- SOF_TIMESTAMPING_TX_SOFTWARE
 - FreeRTOS_TSN_Sockets.h, [140](#)
- source/FreeRTOS_TSN_Ancillary.c, [31](#)
- source/FreeRTOS_TSN_Controller.c, [38](#)
- source/FreeRTOS_TSN_DS.c, [45](#)
- source/FreeRTOS_TSN_NetworkScheduler.c, [48](#)
- source/FreeRTOS_TSN_NetworkSchedulerBlock.c, [55](#)
- source/FreeRTOS_TSN_NetworkSchedulerQueue.c, [61](#)
- source/FreeRTOS_TSN_Sockets.c, [64](#)
- source/FreeRTOS_TSN_Timebase.c, [74](#)
- source/FreeRTOS_TSN_Timestamp.c, [80](#)
- source/FreeRTOS_TSN_VLANTags.c, [82](#)
- source/include/FreeRTOS_TSN_Ancillary.h, [98](#), [107](#)
- source/include/FreeRTOS_TSN_Controller.h, [109](#), [113](#)
- source/include/FreeRTOS_TSN_DS.h, [113](#), [117](#)
- source/include/FreeRTOS_TSN_NetworkScheduler.h, [118](#), [124](#)
- source/include/FreeRTOS_TSN_NetworkSchedulerBlock.h, [125](#), [130](#)
- source/include/FreeRTOS_TSN_NetworkSchedulerQueue.h, [131](#), [135](#)
- source/include/FreeRTOS_TSN_Sockets.h, [136](#), [146](#)
- source/include/FreeRTOS_TSN_Timebase.h, [148](#), [156](#)
- source/include/FreeRTOS_TSN_Timestamp.h, [157](#), [160](#)
- source/include/FreeRTOS_TSN_VLANTags.h, [160](#), [175](#)

- source/include/FreeRTOSConfigDefaults.h, 176, 179
- source/modules/BasicSchedulers/BasicSchedulers.c, 180
- source/modules/BasicSchedulers/BasicSchedulers.h, 183, 184
- source/modules/CreditBasedScheduler/SchedCBS.c, 185
- source/modules/CreditBasedScheduler/SchedCBS.h, 187, 189
- source/portable/NetworkInterface/Common/NetworkWrapper.c, 189
- source/portable/NetworkInterface/include/NetworkWrapper.h, 200, 207
- struct, 13
 - usFrameType, 13
 - xDestinationAddress, 13
 - xSourceAddress, 14
 - xVLANTag, 14
 - xVLANSTag, 14
- TaggedEthernetHeader_t
 - FreeRTOS_TSN_VLANTags.h, 166
- TimeBaseAdjTimeFunction_t
 - FreeRTOS_TSN_Timebase.h, 150
- TimeBaseGetTimeFunction_t
 - FreeRTOS_TSN_Timebase.h, 150
- TimebaseHandle_t
 - FreeRTOS_TSN_Timebase.h, 150
- TimeBaseSetTimeFunction_t
 - FreeRTOS_TSN_Timebase.h, 150
- TimeBaseStartFunction_t
 - FreeRTOS_TSN_Timebase.h, 151
- TimeBaseStopFunction_t
 - FreeRTOS_TSN_Timebase.h, 151
- ts
 - freertos_scm_timestamping, 8
- tsnconfigCONTROLLER_HAS_DYNAMIC_PRIO
 - FreeRTOSConfigDefaults.h, 177
- tsnconfigCONTROLLER_MAX_EVENT_WAIT
 - FreeRTOSConfigDefaults.h, 177
- tsnconfigDEFAULT_QUEUE_TIMEOUT
 - FreeRTOSConfigDefaults.h, 177
- tsnconfigDISABLE
 - FreeRTOSConfigDefaults.h, 177
- tsnconfigDUMP_PACKETS
 - FreeRTOSConfigDefaults.h, 177
- tsnconfigENABLE
 - FreeRTOSConfigDefaults.h, 177
- tsnconfigERRQUEUE_LENGTH
 - FreeRTOSConfigDefaults.h, 178
- tsnconfigINCLUDE_QUEUE_EVENT_CALLBACKS
 - FreeRTOSConfigDefaults.h, 178
- tsnconfigMAX_QUEUE_NAME_LEN
 - FreeRTOSConfigDefaults.h, 178
- tsnconfigSOCKET_INSERTS_VLAN_TAGS
 - FreeRTOSConfigDefaults.h, 178
- tsnconfigTSN_CONTROLLER_PRIORITY
 - FreeRTOSConfigDefaults.h, 178
- tsnconfigWRAPPER_INSERTS_VLAN_TAGS
 - FreeRTOSConfigDefaults.h, 178
- TSNSocket_t
 - FreeRTOS_TSN_Sockets.h, 140
- tsnsocketGET_SOCKET_PORT
 - FreeRTOS_TSN_Sockets.c, 65
- tsnsocketSET_SOCKET_PORT
 - FreeRTOS_TSN_Sockets.c, 66
- tsnsocketSOCKET_IS_BOUND
 - FreeRTOS_TSN_Sockets.c, 66
- tsnsocketSOCKET_TIMEOUT
 - freertos_timespec, 9
- tsnsocketSOCKET_TIMEOUT_SEC
 - freertos_timespec, 9
- ucAttributes
 - xSCHEDULER_GENERIC, 24
- ucDSCClass
 - xTSN_SOCKET, 27
- ucDSCClassGet
 - FreeRTOS_TSN_DS.c, 46
 - FreeRTOS_TSN_DS.h, 116
- ucGetNumberOfTags
 - FreeRTOS_TSN_VLANTags.c, 87
 - FreeRTOS_TSN_VLANTags.h, 166
- ucNumChildren
 - xNETQUEUE_NODE, 18
- uITSFlags
 - xTSN_SOCKET, 28
- usFrameType
 - FreeRTOS_TSN_VLANTags.h, 174
 - struct, 13
- usServiceVLANTag
 - xNETWORK_INTERFACE_CONFIG, 19
- usSize
 - xSCHEDULER_GENERIC, 24
- usTCI
 - xVLAN_TAG, 29
- usTPID
 - xVLAN_TAG, 29
- usVLANTag
 - xNETWORK_INTERFACE_CONFIG, 19
- uxBandwidth
 - xSCHEDULER_CBS, 21
- uxIPV
 - xNETQUEUE, 15
- uxMaxCredit
 - xSCHEDULER_CBS, 21
- uxNetworkQueueGetTicksUntilWakeup
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, 59
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, 129
- uxNetworkQueuePacketsWaiting
 - FreeRTOS_TSN_NetworkSchedulerQueue.c, 63
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, 134
- uxNextActivation
 - xSCHEDULER_CBS, 21
- uxNextWakeup
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, 61
- uxNumQueues

- FreeRTOS_TSN_NetworkScheduler.c, 54
- vAncillaryMsgFree
 - FreeRTOS_TSN_Ancillary.c, 33
 - FreeRTOS_TSN_Ancillary.h, 103
- vAncillaryMsgFreeAll
 - FreeRTOS_TSN_Ancillary.c, 33
 - FreeRTOS_TSN_Ancillary.h, 103
- vAncillaryMsgFreeControl
 - FreeRTOS_TSN_Ancillary.c, 34
 - FreeRTOS_TSN_Ancillary.h, 104
- vAncillaryMsgFreeName
 - FreeRTOS_TSN_Ancillary.c, 34
 - FreeRTOS_TSN_Ancillary.h, 105
- vAncillaryMsgFreePayload
 - FreeRTOS_TSN_Ancillary.c, 35
 - FreeRTOS_TSN_Ancillary.h, 105
- vInitialiseTSNSockets
 - FreeRTOS_TSN_Sockets.c, 73
 - FreeRTOS_TSN_Sockets.h, 145
- vlantagCLASS_0
 - FreeRTOS_TSN_VLANTags.h, 162
- vlantagCLASS_1
 - FreeRTOS_TSN_VLANTags.h, 162
- vlantagCLASS_2
 - FreeRTOS_TSN_VLANTags.h, 163
- vlantagCLASS_3
 - FreeRTOS_TSN_VLANTags.h, 163
- vlantagCLASS_4
 - FreeRTOS_TSN_VLANTags.h, 163
- vlantagCLASS_5
 - FreeRTOS_TSN_VLANTags.h, 163
- vlantagCLASS_6
 - FreeRTOS_TSN_VLANTags.h, 163
- vlantagCLASS_7
 - FreeRTOS_TSN_VLANTags.h, 163
- vlantagDEI_BIT_MASK
 - FreeRTOS_TSN_VLANTags.h, 163
- vlantagETH_TAG_OFFSET
 - FreeRTOS_TSN_VLANTags.h, 163
- vlantagGET_DEI_FROM_TCI
 - FreeRTOS_TSN_VLANTags.h, 164
- vlantagGET_PCP_FROM_TCI
 - FreeRTOS_TSN_VLANTags.h, 164
- vlantagGET_VID_FROM_TCI
 - FreeRTOS_TSN_VLANTags.h, 164
- vlantagPCP_BIT_MASK
 - FreeRTOS_TSN_VLANTags.h, 164
- vlantagSET_DEI_FROM_TCI
 - FreeRTOS_TSN_VLANTags.h, 164
- vlantagSET_PCP_FROM_TCI
 - FreeRTOS_TSN_VLANTags.h, 164
- vlantagSET_VID_FROM_TCI
 - FreeRTOS_TSN_VLANTags.h, 165
- vlantagTPID_DEFAULT
 - FreeRTOS_TSN_VLANTags.h, 165
- vlantagTPID_DOUBLE_TAG
 - FreeRTOS_TSN_VLANTags.h, 165
- vlantagVID_BIT_MASK
 - FreeRTOS_TSN_VLANTags.h, 165
- FreeRTOS_TSN_VLANTags.h, 165
- vNetworkQueueAddWakeupEvent
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, 59
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, 129
- vNetworkQueueInit
 - FreeRTOS_TSN_NetworkScheduler.h, 119
 - NetworkWrapper.c, 195
- vNetworkQueueListAdd
 - FreeRTOS_TSN_NetworkScheduler.c, 49
 - FreeRTOS_TSN_NetworkScheduler.h, 120
- vRetrieveHardwareTimestamp
 - NetworkWrapper.c, 196
 - NetworkWrapper.h, 202
- vSocketFromPort
 - FreeRTOS_TSN_Sockets.c, 73
 - FreeRTOS_TSN_Sockets.h, 146
- vTimebaseAdjTime
 - FreeRTOS_TSN_Timebase.c, 76
 - FreeRTOS_TSN_Timebase.h, 151
- vTimebaseGetTime
 - FreeRTOS_TSN_Timebase.c, 76
 - FreeRTOS_TSN_Timebase.h, 151
- vTimebaseInit
 - FreeRTOS_TSN_Timebase.h, 153
 - NetworkWrapper.c, 196
- vTimebaseSetTime
 - FreeRTOS_TSN_Timebase.c, 76
 - FreeRTOS_TSN_Timebase.h, 153
- vTimebaseStart
 - FreeRTOS_TSN_Timebase.c, 76
 - FreeRTOS_TSN_Timebase.h, 153
- vTimebaseStop
 - FreeRTOS_TSN_Timebase.h, 153
- vTimestampAcquireSoftware
 - FreeRTOS_TSN_Timestamp.c, 81
 - FreeRTOS_TSN_Timestamp.h, 158
- vTSNController_Initialise
 - FreeRTOS_TSN_Controller.c, 42
 - FreeRTOS_TSN_Controller.h, 110
- vTSNControllerComputePriority
 - FreeRTOS_TSN_Controller.c, 42
 - FreeRTOS_TSN_Controller.h, 110
- wrapperFIRST_TPID
 - NetworkWrapper.c, 190
- wrapperSECOND_TPID
 - NetworkWrapper.c, 191
- xAncillaryMsgControlFill
 - FreeRTOS_TSN_Ancillary.c, 35
 - FreeRTOS_TSN_Ancillary.h, 105
- xAncillaryMsgControlFillSingle
 - FreeRTOS_TSN_Ancillary.c, 36
 - FreeRTOS_TSN_Ancillary.h, 106
- xAncillaryMsgFillName
 - FreeRTOS_TSN_Ancillary.c, 36
 - FreeRTOS_TSN_Ancillary.h, 106
- xAncillaryMsgFillPayload
 - FreeRTOS_TSN_Ancillary.c, 37

- FreeRTOS_TSN_Ancillary.h, 107
- xBaseSocket
 - xTSN_SOCKET, 28
- xBoundSocketListItem
 - xTSN_SOCKET, 28
- xDestinationAddress
 - FreeRTOS_TSN_VLANTags.h, 174
 - struct, 13
- xDSClassSet
 - FreeRTOS_TSN_DS.c, 47
 - FreeRTOS_TSN_DS.h, 116
- xEMACIndex
 - xNETWORK_INTERFACE_CONFIG, 19
- xErrQueue
 - xTSN_SOCKET, 28
- xGetPhyLinkStatus
 - NetworkWrapper.c, 191, 196
- xisCallingFromTSNController
 - FreeRTOS_TSN_Controller.c, 43
 - FreeRTOS_TSN_Controller.h, 111
- xMAC_GetPhyLinkStatus
 - NetworkWrapper.h, 203
- xMAC_NetworkInterfaceInitialise
 - NetworkWrapper.h, 203
- xMAC_NetworkInterfaceOutput
 - NetworkWrapper.h, 203
- xNETQUEUE, 14
 - cName, 15
 - ePolicy, 15
 - fnFilter, 15
 - uxIPV, 15
 - xQueue, 15
- xNETQUEUE_ITEM, 16
 - eEventType, 17
 - pxBuf, 17
 - pxMsg, 17
 - xReleaseAfterSend, 17
- xNETQUEUE_NODE, 17
 - pvScheduler, 18
 - pxNext, 18
 - pxQueue, 18
 - ucNumChildren, 18
- xNETWORK_INTERFACE_CONFIG, 19
 - usServiceVLANTag, 19
 - usVLANTag, 19
 - xEMACIndex, 19
 - xNumTags, 19
- xNetworkInterfaceInitialise
 - NetworkWrapper.c, 191, 197
- xNetworkInterfaceOutput
 - NetworkWrapper.c, 191, 197
- xNetworkQueueAssignRoot
 - FreeRTOS_TSN_NetworkScheduler.c, 50
 - FreeRTOS_TSN_NetworkScheduler.h, 120
- xNetworkQueueInsertPacketByFilter
 - FreeRTOS_TSN_NetworkScheduler.c, 50
 - FreeRTOS_TSN_NetworkScheduler.h, 120
- xNetworkQueueInsertPacketByName
 - FreeRTOS_TSN_NetworkScheduler.c, 51
 - FreeRTOS_TSN_NetworkScheduler.h, 121
- xNetworkQueueIsEmpty
 - FreeRTOS_TSN_NetworkSchedulerQueue.c, 63
 - FreeRTOS_TSN_NetworkSchedulerQueue.h, 134
- xNetworkQueuePop
 - FreeRTOS_TSN_NetworkScheduler.c, 52
 - FreeRTOS_TSN_NetworkScheduler.h, 122
- xNetworkQueuePush
 - FreeRTOS_TSN_NetworkScheduler.c, 52
 - FreeRTOS_TSN_NetworkScheduler.h, 122
- xNetworkQueueSchedule
 - FreeRTOS_TSN_NetworkScheduler.c, 53
 - FreeRTOS_TSN_NetworkScheduler.h, 123
- xNetworkSchedulerLinkChild
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, 60
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, 129
- xNetworkSchedulerLinkQueue
 - FreeRTOS_TSN_NetworkSchedulerBlock.c, 60
 - FreeRTOS_TSN_NetworkSchedulerBlock.h, 130
- xNetworkWrapperInitialised
 - NetworkWrapper.c, 200
- xNotifyController
 - FreeRTOS_TSN_Controller.c, 43
 - FreeRTOS_TSN_Controller.h, 111
- xNumTags
 - xNETWORK_INTERFACE_CONFIG, 19
- xQueue
 - xNETQUEUE, 15
- xQUEUE_LIST, 20
 - pxNext, 20
 - pxQueue, 20
- xRecvTask
 - xTSN_SOCKET, 28
- xReleaseAfterSend
 - xNETQUEUE_ITEM, 17
- xScheduler
 - xSCHEDULER_CBS, 22
 - xSCHEDULER_FIFO, 22
 - xSCHEDULER_PRIO, 25
 - xSCHEDULER_RR, 25
- xSCHEDULER_CBS, 21
 - uxBandwidth, 21
 - uxMaxCredit, 21
 - uxNextActivation, 21
 - xScheduler, 22
- xSCHEDULER_FIFO, 22
 - xScheduler, 22
- xSCHEDULER_GENERIC, 23
 - fnReady, 23
 - fnSelect, 24
 - pxOwner, 24
 - ucAttributes, 24
 - usSize, 24
- xSCHEDULER_PRIO, 24
 - xScheduler, 25
- xSCHEDULER_RR, 25
 - xScheduler, 25

- xSendEventStructToIPTask
 - NetworkWrapper.c, [191](#)
- xSendEventStructToTSNController
 - NetworkWrapper.c, [197](#)
 - NetworkWrapper.h, [204](#)
- xSendTask
 - xTSN_SOCKET, [28](#)
- xSocketErrorQueueInsert
 - FreeRTOS_TSN_Sockets.c, [73](#)
 - FreeRTOS_TSN_Sockets.h, [146](#)
- xSourceAddress
 - FreeRTOS_TSN_VLANTags.h, [174](#)
 - struct, [14](#)
- xTIMEBASE, [26](#)
 - fnAdjTime, [26](#)
 - fnGetTime, [26](#)
 - fnSetTime, [27](#)
 - fnStart, [27](#)
 - fnStop, [27](#)
- xTimebaseGetState
 - FreeRTOS_TSN_Timebase.c, [77](#)
 - FreeRTOS_TSN_Timebase.h, [154](#)
- xTimebaseHandle
 - FreeRTOS_TSN_Timebase.c, [80](#)
- xTimebaseHandleSet
 - FreeRTOS_TSN_Timebase.c, [77](#)
 - FreeRTOS_TSN_Timebase.h, [154](#)
- xTimebaseState
 - FreeRTOS_TSN_Timebase.c, [80](#)
- xTimespecCmp
 - FreeRTOS_TSN_Timebase.c, [77](#)
 - FreeRTOS_TSN_Timebase.h, [154](#)
- xTimespecDiff
 - FreeRTOS_TSN_Timebase.c, [78](#)
 - FreeRTOS_TSN_Timebase.h, [155](#)
- xTimespecDiv
 - FreeRTOS_TSN_Timebase.c, [78](#)
 - FreeRTOS_TSN_Timebase.h, [155](#)
- xTimespecSum
 - FreeRTOS_TSN_Timebase.c, [79](#)
 - FreeRTOS_TSN_Timebase.h, [156](#)
- xTSN_GetPhyLinkStatus
 - NetworkWrapper.c, [198](#)
 - NetworkWrapper.h, [205](#)
- xTSN_NetworkInterfaceInitialise
 - NetworkWrapper.c, [199](#)
 - NetworkWrapper.h, [205](#)
- xTSN_NetworkInterfaceOutput
 - NetworkWrapper.c, [199](#)
 - NetworkWrapper.h, [206](#)
- xTSN_SOCKET, [27](#)
 - ucDSClass, [27](#)
 - ulTSFlags, [28](#)
 - xBaseSocket, [28](#)
 - xBoundSocketListItem, [28](#)
 - xErrQueue, [28](#)
 - xRecvTask, [28](#)
 - xSendTask, [28](#)
- xTSNBoundUDPSocketList
 - FreeRTOS_TSN_Sockets.c, [74](#)
- xTSNControllerHandle
 - FreeRTOS_TSN_Controller.c, [45](#)
- xTSNControllerUpdatePriority
 - FreeRTOS_TSN_Controller.c, [44](#)
 - FreeRTOS_TSN_Controller.h, [112](#)
- xVLAN_TAG, [29](#)
 - usTCI, [29](#)
 - usTPID, [29](#)
- xVLANCTag
 - struct, [14](#)
- xVLANCTagCheckClass
 - FreeRTOS_TSN_VLANTags.c, [88](#)
- xVLANCTagGetDEI
 - FreeRTOS_TSN_VLANTags.c, [89](#)
- xVLANCTagGetPCP
 - FreeRTOS_TSN_VLANTags.c, [89](#)
- xVLANCTagGetVID
 - FreeRTOS_TSN_VLANTags.c, [90](#)
- xVLANCTagSetDEI
 - FreeRTOS_TSN_VLANTags.c, [91](#)
 - FreeRTOS_TSN_VLANTags.h, [167](#)
- xVLANCTagSetPCP
 - FreeRTOS_TSN_VLANTags.c, [92](#)
 - FreeRTOS_TSN_VLANTags.h, [168](#)
- xVLANCTagSetVID
 - FreeRTOS_TSN_VLANTags.c, [92](#)
 - FreeRTOS_TSN_VLANTags.h, [168](#)
- xVLANSTag
 - struct, [14](#)
- xVLANSTagCheckClass
 - FreeRTOS_TSN_VLANTags.c, [93](#)
 - FreeRTOS_TSN_VLANTags.h, [169](#)
- xVLANSTagGetDEI
 - FreeRTOS_TSN_VLANTags.c, [94](#)
 - FreeRTOS_TSN_VLANTags.h, [170](#)
- xVLANSTagGetPCP
 - FreeRTOS_TSN_VLANTags.c, [94](#)
 - FreeRTOS_TSN_VLANTags.h, [170](#)
- xVLANSTagGetVID
 - FreeRTOS_TSN_VLANTags.c, [95](#)
 - FreeRTOS_TSN_VLANTags.h, [171](#)
- xVLANSTagSetDEI
 - FreeRTOS_TSN_VLANTags.c, [96](#)
 - FreeRTOS_TSN_VLANTags.h, [172](#)
- xVLANSTagSetPCP
 - FreeRTOS_TSN_VLANTags.c, [97](#)
 - FreeRTOS_TSN_VLANTags.h, [173](#)
- xVLANSTagSetVID
 - FreeRTOS_TSN_VLANTags.c, [97](#)
 - FreeRTOS_TSN_VLANTags.h, [173](#)
- xVLANTag
 - FreeRTOS_TSN_VLANTags.h, [175](#)
- xVLANTagCheckClass
 - FreeRTOS_TSN_VLANTags.h, [165](#)
- xVLANTagGetDEI
 - FreeRTOS_TSN_VLANTags.h, [165](#)

xVLANTagGetPCP
FreeRTOS_TSN_VLANTags.h, [166](#)
xVLANTagGetVID
FreeRTOS_TSN_VLANTags.h, [166](#)
xVLANTagSetDEI
FreeRTOS_TSN_VLANTags.h, [166](#)
xVLANTagSetPCP
FreeRTOS_TSN_VLANTags.h, [166](#)
xVLANTagSetVID
FreeRTOS_TSN_VLANTags.h, [166](#)