TCNOpen TRDP

Prototype

Generated by Doxygen 1.5.6

Mon Jun 4 15:28:49 2012

Contents

Chapter 1

The TRDP Light Library API Specification

1.1 General Information

1.1.1 Purpose

The TRDP protocol has been defined as the standard communication protocol in IP-enabled trains. It allows communication via process data (periodically transmitted data using UDP/IP) and message data (client - server messaging using UDP/IP or TCP/IP) This document describes the light API of the TRDP Library.

1.1.2 Scope

The intended audience of this document is the developers and project members of the TRDP project. TRDP Client Applications are programs using the TRDP protocol library to access the services of TRDP. Programmers developing such applications are the main target audience for this documentation.

1.1.3 Related documents

TCN-TRDP2-D-BOM-004-01 IEC61375-2-3_CD_ANNEXA Protocol definition of the TRDP standard

1.1.4 Abbreviations and Definitions

-*API* Application Programming Interface

-*ECN* Ethernet Consist Network

-*TRDP* Train Real-time Data Protocol

-*TCMS* Train Control Management System

1.2 Terminology

The API documented here is mainly concerned with three bodies of code:

- *TRDP Client Applications* (or 'client applications' for short): These are programs using the API to access the services of TRDP. Programmers developing such applications are the main target audience for this documentation.

- *TRDP Light Implementations* (or just 'TRDP implementation'): These are libraries realising the API as documented here. Programmers developing such implementations will find useful definitions about syntax and semantics of the API wihtin this documentation.

- *VOS Subsystem* (Virtual Operating System): An OS and hardware abstraction layer which offers memory, networking, threading, queues and debug functions. The VOS API is documented here.

The following diagram shows how these pieces of software are interrelated.

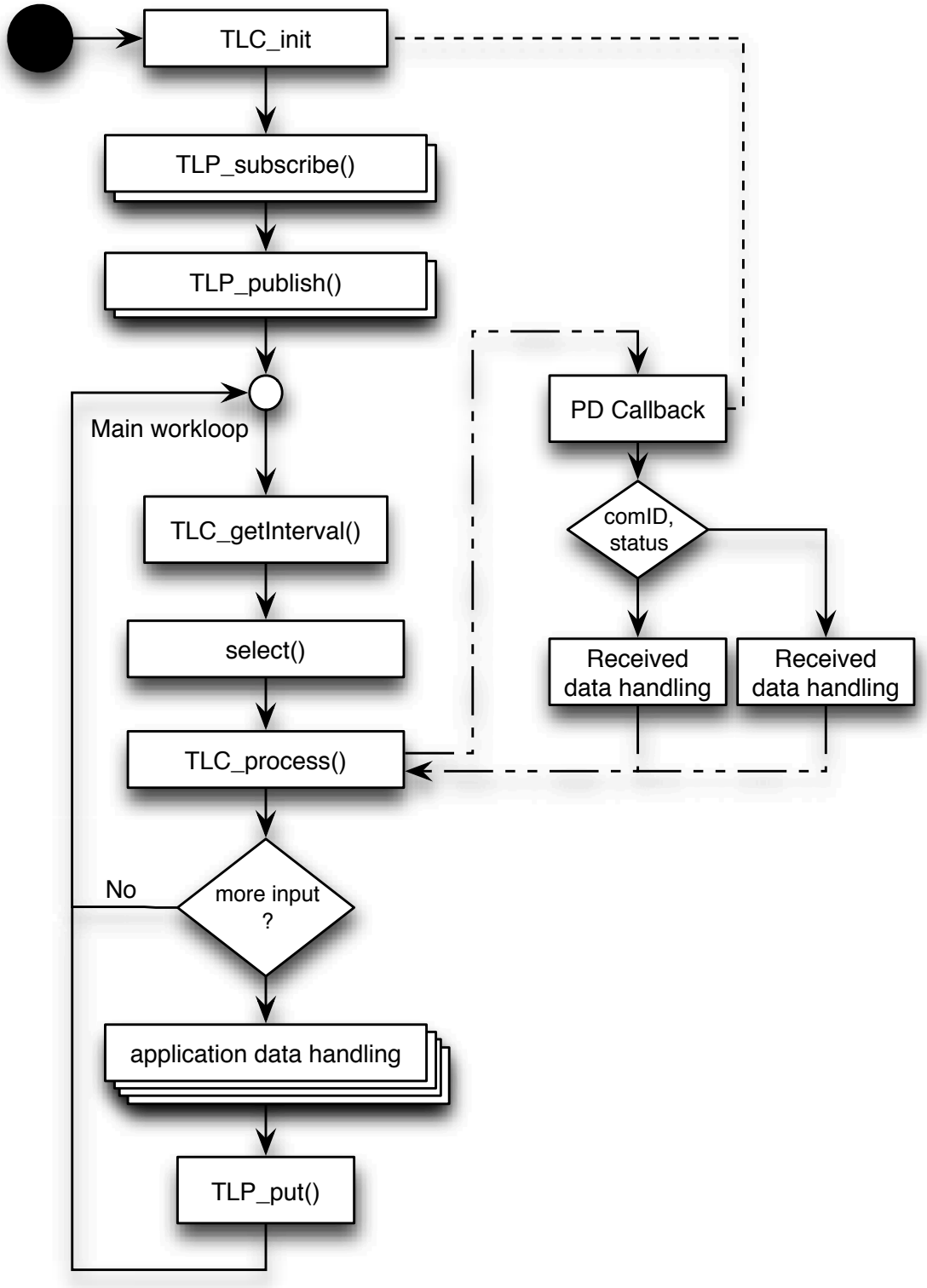


Figure 1.1: Sample client workflow

1.3 Conventions of the API

The API comprises a set of C header files that can also be used from client applications written in C++. These header files are contained in a directory named `trdp/api` and a subdirectory called `trdp/vos/api` with declarations not topical to TRDP but needed by the stack. Client applications shall include these header files like:

```
#include "trdp_if_light.h"
```

and, if VOS functions are needed, also the corresponding headers:

```
#include "vos_thread.h"
```

for example.

The subdirectory `trdp/doc` contains files needed for the API documentation.

Generally client application source code including API headers will only compile if the parent directory of the `trdp` directory is part of the include path of the used compiler. No other subdirectories of the API should be added to the compiler's include path.

The client API doesn't support a "catch-all" header file that includes all declarations in one step; rather the client application has to include individual headers for each feature set it wants to use.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 __attribute__ Struct Reference

TRDP process data header - network order and alignment.

`#include <trdp_private.h>`

Data Fields

- UINT32 sequenceCounter
  Unique counter (autom incremented).
- UINT16 protocolVersion
  fix value for compatibility (set by the API)
- UINT16 msgType
  of datagram: PD Request (0x5072) or PD_MSG (0x5064)
- UINT32 comId
  set by user: unique id
- UINT32 topoCount
  set by user: ETB to use, '0' to deacticate
- UINT32 datasetLength
  length of the data to transmit 0.
- UINT16 subsAndReserved
  first bit (MSB): indicates substitution transmission
- UINT16 offsetAddress
  for process data in traffic store
- UINT32 replyComId
  used in PD request

- UINT32 replyIpAddress
  used for PD request
- INT32 replyStatus
  0 = OK
- UINT8 sessionID [16]
  UUID as a byte stream.
- UINT32 replyTimeout
  in us
- UINT8 sourceURI [32]
  User part of URI.
- UINT8 destinationURI [32]
  User part of URI.

4.1.1 Detailed Description

TRDP process data header - network order and alignment.

TRDP message data header - network order and alignment.

4.1.2 Field Documentation

4.1.2.1 UINT16 __attribute__::protocolVersion

fix value for compatibility (set by the API)

fix value for compatibility

4.1.2.2 UINT16 __attribute__::msgType

of datagram: PD Request (0x5072) or PD_MSG (0x5064)

of datagram: Mn, Mr, Mp, Mq, Mc or Me

4.1.2.3 UINT32 __attribute__::datasetLength

length of the data to transmit 0.

defined by user: length of data to transmit

..1436 without padding and FCS

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

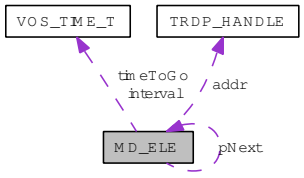
- trdp_private.h

4.2 MD_ELE Struct Reference

Queue element for MD packets to send or receive or acknowledge.

`#include <trdp_private.h>`

Collaboration diagram for MD_ELE:



Data Fields

- struct MD_ELE * pNext
  pointer to next element or NULL
- TRDP_ADDRESSES addr
  handle of publisher/subscriber
- TRDP_PRIV_FLAGS_T privFlags
  private flags
- TRDP_TIME_T interval
  time out value for received packets or interval for packets to send (set from ms)
- TRDP_TIME_T timeToGo
  next time this packet must be sent/rcv
- INT32 dataSize
  net data size
- INT32 socketIdx
  index into the socket list
- MD_HEADER_T frameHead
  Packet header in network byte order.
- UINT8 data [0]
  data ready to be sent (with CRCs)

4.2.1 Detailed Description

Queue element for MD packets to send or receive or acknowledge.

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

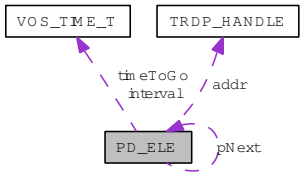
- trdp_private.h

4.3 PD_ELE Struct Reference

Queue element for PD packets to send or receive.

`#include <trdp_private.h>`

Collaboration diagram for PD_ELE:



Data Fields

- struct PD_ELE * pNext
  pointer to next element or NULL
- TRDP_ADDRESSES addr
  handle of publisher/subscriber
- TRDP_PRIV_FLAGS_T privFlags
  private flags
- TRDP_FLAGS_T pktFlags
  flags
- TRDP_TIME_T interval
  time out value for received packets or interval for packets to send (set from ms)
- TRDP_TIME_T timeToGo
  next time this packet must be sent/rcv
- INT32 dataSize
  net data size
- INT32 grossSize
  complete packet size (header, data, padding, FCS)
- INT32 socketIdx
  index into the socket list
- const void * userRef
  from subscribe()
- PD_HEADER_T frameHead
  Packet header in network byte order.

- UINT8 data [MAX_PD_PACKET_SIZE]
  data ready to be sent or received (with CRCs)

4.3.1 Detailed Description

Queue element for PD packets to send or receive.

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

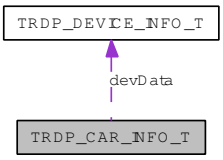
- trdp_private.h

4.4 TRDP_CAR_INFO_T Struct Reference

car information structure.

`#include <tau_addr.h>`

Collaboration diagram for TRDP_CAR_INFO_T:



Data Fields

- TRDP_LABEL_T id
  Unique car identifier (Label) / UIC identification nuber.
- TRDP_LABEL_T type
  Car type.
- UINT8 cstCarNo
  sequence number of car in consist
- UINT8 trnOrient
  opposite(0) or same(1) orientation rel.
- UINT8 cstOrient
  opposite(0) or same(1) orientation rel.
- UINT8 reserved1
  reserved for alignment and future use
- UINT16 devCnt
  number of devices in the car
- TRDP_DEVICE_INFO_T devData [1]
  device data list.

4.4.1 Detailed Description

car information structure.

4.4.2 Field Documentation

4.4.2.1 UINT8 TRDP_CAR_INFO_T::trnOrient

opposite(0) or same(1) orientation rel.

to train

4.4.2.2 UINT8 TRDP_CAR_INFO_T::cstOrient

opposite(0) or same(1) orientation rel.

to consist

4.4.2.3 TRDP_DEVICE_INFO_T TRDP_CAR_INFO_T::devData[1]

device data list.

The list size '1' is just a proxy definition for the real size (devCnt) in order to satisfy C-Language

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

- tau_addr.h

4.5 TRDP_DATASET_ELEMENT_T Struct Reference

Dataset element definition.

`#include <trdp_types.h>`

Data Fields

- INT32 type

  Data type or dataset id.

- UINT32 size

  Number of items or TDRP_VAR_SIZE (0).

4.5.1 Detailed Description

Dataset element definition.

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

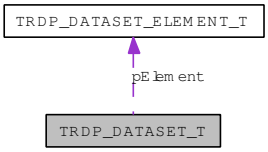
- trdp_types.h

4.6 TRDP_DATASET_T Struct Reference

Dataset definition.

`#include <trdp_types.h>`

Collaboration diagram for TRDP_DATASET_T:



Data Fields

- INT32 id
  dataset identifier
- UINT16 reserved1
  Reserved for future use, must be zero.
- UINT16 numElement
  Number of elements.
- TRDP_DATASET_ELEMENT_T * pElement
  Pointer to a dataset element, used as array.

4.6.1 Detailed Description

Dataset definition.

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

- trdp_types.h

4.7 TRDP_DBG_CONFIG_T Struct Reference

Control for debug output device/file on application level.

`#include <tau_xml.h>`

Data Fields

- TRDP_DEBUG_OPTION_T option
  Debug printout options for application use.
- UINT32 maxFileSize
  Maximal file size.
- TRDP_FILE_NAME_T fileName
  Debug file name and path.

4.7.1 Detailed Description

Control for debug output device/file on application level.

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

- tau_xml.h

4.8 TRDP_DEVICE_INFO_T Struct Reference

device information structure

`#include <tau_addr.h>`

Data Fields

- TRDP_IP_ADDR ipAddr
  device IP address
- TRDP_LABEL_T id
  device identifier (Label) / host name
- TRDP_LABEL_T type
  device type
- UINT8 orient
  device orientation 0=opposite, 1=same rel.
- UINT8 reserved1
  reserved for alignment and future use
- UINT16 length
  length related to car
- UINT16 width
  width related to car
- UINT16 hight
  hight related to car

4.8.1 Detailed Description

device information structure

4.8.2 Field Documentation

4.8.2.1 UINT8 TRDP_DEVICE_INFO_T::orient

device orientation 0=opposite, 1=same rel.

to car

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

- tau_addr.h

4.9 TRDP_HANDLE Struct Reference

Hidden handle definition, used as unique addressing item.

`#include <trdp_private.h>`

Data Fields

- UINT32 comId
  comId for packets to send/receive
- TRDP_IP_ADDR_T srcIpAddr
  source IP for PD
- TRDP_IP_ADDR_T destIpAddr
  destination IP for PD
- TRDP_IP_ADDR_T mcGroup
  multicast group to join for PD

4.9.1 Detailed Description

Hidden handle definition, used as unique addressing item.

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

- trdp_private.h

4.10 TRDP_LIST_STATISTICS_T Struct Reference

Information about a particular MD listener.

`#include <trdp_types.h>`

Data Fields

- UINT32 comId
  ComId to listen to.
- TRDP_URI_USER_T uri
  URI user part to listen to.
- TRDP_IP_ADDR_T joinedAddr
  Joined IP address.
- UINT32 callBack
  Call back function reference if used.
- UINT32 queue
  Queue reference if used.
- UINT32 userRef
  User reference if used.
- UINT32 numRecv
  Number of received packets.

4.10.1 Detailed Description

Information about a particular MD listener.

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

- trdp_types.h

4.11 TRDP_MARSHALL_CONFIG_T Struct Reference

Marshaling/unmarshalling configuration.

`#include <trdp_types.h>`

Data Fields

- TRDP_MARSHALL_T pCbMarshall

  Pointer to marshall callback function.

- TRDP_UNMARSHALL_T pCbUnmarshall

  Pointer to unmarshall callback function.

- void * pRefCon

  Pointer to user context for call back.

4.11.1 Detailed Description

Marshaling/unmarshalling configuration.

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

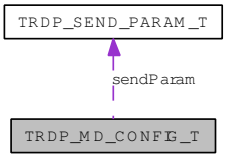
- trdp_types.h

4.12 TRDP_MD_CONFIG_T Struct Reference

Default MD configuration.

`#include <trdp_types.h>`

Collaboration diagram for TRDP_MD_CONFIG_T:



Data Fields

- TRDP_MD_CALLBACK_T pCbFunction
  Pointer to MD callback function.
- void * pRefCon
  Pointer to user context for call back.
- TRDP_SEND_PARAM_T sendParam
  Default send parameters.
- TRDP_FLAGS_T flags
  Default flags for MD packets.
- UINT32 replyTimeout
  Default timeout in us.
- UINT32 confirmTimeout
  Default timeout in us.
- UINT32 udpPort
  Port to be used for UDP MD communication.
- UINT32 tcpPort
  Port to be used for TCP MD communication.

4.12.1 Detailed Description

Default MD configuration.

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

- trdp_types.h

4.13 TRDP_MD_INFO_T Struct Reference

Message data info from received telegram; allows the application to generate responses.

`#include <trdp_types.h>`

Data Fields

- TRDP_IP_ADDR_T srcIpAddr
  source IP address for filtering
- TRDP_IP_ADDR_T destIpAddr
  destination IP address for filtering
- UINT32 seqCount
  sequence counter
- UINT16 protVersion
  Protocol version.
- TRDP_MSG_T msgType
  Protocol ('PD', 'MD', .
- UINT32 comId
  ComID.
- UINT32 topoCount
  received topocount
- UINT16 userStatus
  error code, user stat
- TRDP_REPLY_STATUS_T replyStatus
  reply status
- TRDP_UUID_T sessionId
  for response
- UINT32 replyTimeout
  reply timeout in us given with the request
- TRDP_URI_USER_T destURI
  destination URI user part from MD header
- TRDP_URI_USER_T srcURI
  source URI user part from MD header
- UINT32 noOfReplies
  actual number of replies for the request

- const void * pUserRef
  User reference given with the local call.
- TRDP_ERR_T resultCode
  error code

4.13.1 Detailed Description

Message data info from received telegram; allows the application to generate responses.

Note: Not all fields are relevant for each message type!

4.13.2 Field Documentation

4.13.2.1 TRDP_MSG_T TRDP_MD_INFO_T::msgType

Protocol ('PD', 'MD', .

..)

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

- trdp_types.h

4.14 TRDP_MD_STATISTICS Struct Reference

Message data statistics.

`#include <trdp_private.h>`

Data Fields

- UINT32 headerInPackets
  Incoming packets.
- UINT32 headerInCRCErr
  Incoming CRC errors.
- UINT32 headerInProtoErr
  Incoming protocol errors.
- UINT32 headerInTimeOuts
  Incoming timing errors.
- UINT32 headerInFrameErr
  Incoming timing errors.
- UINT32 headerOutPackets
  Outgoing packets.
- UINT32 headerAckErr
  Missing acknowledge.

4.14.1 Detailed Description

Message data statistics.

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

- trdp_private.h

4.15 TRDP_MD_STATISTICS_T Struct Reference

Structure containing all general MD statistics information.

`#include <trdp_types.h>`

Data Fields

- UINT32 defQos

  default QoS for MD

- UINT32 defTtl

  default TTL for MD

- UINT32 defReplyTimeout

  default reply timeout in us for MD

- UINT32 defConfirmTimeout

  default confirm timeout in us for MD

- UINT32 numList

  number of listeners

- UINT32 numRcv

  number of received MD packets

- UINT32 numCrcErr

  number of received MD packets with CRC err

- UINT32 numProtErr

  number of received MD packets with protocol err

- UINT32 numTopoErr

  number of received MD packets with wrong topo count

- UINT32 numNoListener

  number of received MD packets without listener

- UINT32 numReplyTimeout

  number of reply timeouts

- UINT32 numConfirmTimeout

  number of confirm timeouts

- UINT32 numSend

  number of sent MD packets

4.15.1 Detailed Description

Structure containing all general MD statistics information.

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

- trdp_types.h

4.16 TRDP_MEM_CONFIG_T Struct Reference

Structure describing memory (and its pre-fragmentation).

`#include <trdp_types.h>`

Data Fields

- UINT8 * p
  pointer to static or allocated memory
- UINT32 size
  size of static or allocated memory
- UINT32 prealloc [TRDP_MEM_BLK_524288+1]
  memory block structure

4.16.1 Detailed Description

Structure describing memory (and its pre-fragmentation).

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

- trdp_types.h

4.17 TRDP_MEM_STATISTICS_T Struct Reference

TRDP statistics type definitions.

`#include <trdp_types.h>`

Data Fields

- UINT32 total
  total memory size
- UINT32 free
  free memory size
- UINT32 minFree
  minimal free memory size in statistics interval
- UINT32 numAllocBlocks
  allocated memory blocks
- UINT32 numAllocErr
  allocation errors
- UINT32 numFreeErr
  free errors
- UINT32 allocBlockSize [TRDP_MEM_BLK_524288+1]
  allocated memory blocks
- UINT32 usedBlockSize [TRDP_MEM_BLK_524288+1]
  used memory blocks

4.17.1 Detailed Description

TRDP statistics type definitions.

Statistical data regarding the former info provided via SNMP the following information was left out/can be implemented additionally using MD:

- PD subscr table: ComId, sourceIpAddr, destIpAddr, cbFct?, timout, toBehaviour, counter
- PD publish table: ComId, destIpAddr, redId, redState cycle, ttl, qos, counter
- PD join table: joined MC address table
- MD listener table: ComId destIpAddr, destUri, cbFct?, counter
- Memory usage Structure containing all general memory statistics information.

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

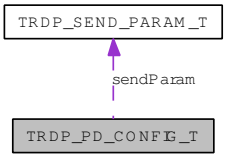
- trdp_types.h

4.18 TRDP_PD_CONFIG_T Struct Reference

Default PD configuration.

`#include <trdp_types.h>`

Collaboration diagram for TRDP_PD_CONFIG_T:



Data Fields

- TRDP_PD_CALLBACK_T pCbFunction
  Pointer to PD callback function.
- void * pRefCon
  Pointer to user context for call back.
- TRDP_SEND_PARAM_T sendParam
  Default send parameters.
- TRDP_FLAGS_T flags
  Default flags for PD packets.
- UINT32 timeout
  Default timeout in us.
- TRDP_TO_BEHAVIOR_T toBehavior
  Default timeout behaviour.
- UINT32 port
  Port to be used for PD communication.

4.18.1 Detailed Description

Default PD configuration.

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

- trdp_types.h

4.19 TRDP_PD_INFO_T Struct Reference

Process data info from received telegram; allows the application to generate responses.

`#include <trdp_types.h>`

Data Fields

- TRDP_IP_ADDR_T srcIpAddr

  source IP address for filtering

- TRDP_IP_ADDR_T destIpAddr

  destination IP address for filtering

- UINT32 seqCount

  sequence counter

- UINT16 protVersion

  Protocol version.

- TRDP_MSG_T msgType

  Protocol ('PD', 'MD', .

- UINT32 comId

  ComID.

- UINT32 topoCount

  received topocount

- BOOL subs

  substitution

- UINT16 offsetAddr

  offset address for ladder architecture

- UINT32 replyComId

  ComID for reply (request only).

- TRDP_IP_ADDR_T replyIpAddr

  IP address for reply (request only).

- const void * pUserRef

  User reference given with the local subscribe.

- TRDP_ERR_T resultCode

  error code

4.19.1 Detailed Description

Process data info from received telegram; allows the application to generate responses.

Note: Not all fields are relevant for each message type!

4.19.2 Field Documentation

4.19.2.1 TRDP_MSG_T TRDP_PD_INFO_T::msgType

Protocol ('PD', 'MD', .

..)

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

- trdp_types.h

4.20 TRDP_PD_STATISTICS Struct Reference

Process data statistics.

`#include <trdp_private.h>`

Data Fields

- UINT32 headerInPackets
  Incoming packets.
- UINT32 headerInCRCErr
  Incoming CRC errors.
- UINT32 headerInProtoErr
  Incoming protocol errors.
- UINT32 headerInTimeOuts
  Incoming timing errors.
- UINT32 headerInFrameErr
  Incoming timing errors.
- UINT32 headerOutPackets
  Outgoing packets.

4.20.1 Detailed Description

Process data statistics.

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

- trdp_private.h

4.21 TRDP_PD_STATISTICS_T Struct Reference

Structure containing all general PD statistics information.

`#include <trdp_types.h>`

Data Fields

- UINT32 defQos

  default QoS for PD

- UINT32 defTtl

  default TTL for PD

- UINT32 defTimeout

  default timeout in us for PD

- UINT32 numSubs

  number of subscribed ComId's

- UINT32 numPub

  number of published ComId's

- UINT32 numRcv

  number of received PD packets

- UINT32 numCrcErr

  number of received PD packets with CRC err

- UINT32 numProtErr

  number of received PD packets with protocol err

- UINT32 numTopoErr

  number of received PD packets with wrong topo count

- UINT32 numNoSubs

  number of received PD push packets without subscription

- UINT32 numNoPub

  number of received PD pull packets without publisher

- UINT32 numTimeout

  number of PD timeouts

- UINT32 numSend

  number of sent PD packets

4.21.1 Detailed Description

Structure containing all general PD statistics information.

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

- trdp_types.h

4.22 TRDP_PROCESS_CONFIG_T Struct Reference

Types to read out the XML configuration.

`#include <tau_xml.h>`

Data Fields

- TRDP_LABEL_T hostName

  Host name.

- TRDP_LABEL_T leaderName

  Leader name dependant on redundanca concept.

- TRDP_IP_ADDR hostIp

  Host IP address.

- TRDP_IP_ADDR leaderIp

  Leader IP address dependant on redundancy concept.

- UINT32 cycleTime

  TRDP main process cycle time in usec.

- UINT32 priority

  TRDP main process priority.

- TRDP_OPTION_T options

  TRDP default options.

4.22.1 Detailed Description

Types to read out the XML configuration.

Configuration of TRDP main process.

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

- tau_xml.h

4.23 TRDP_PUB_STATISTICS_T Struct Reference

Table containing particular PD publishing information.

`#include <trdp_types.h>`

Data Fields

- UINT32 comId
  Published ComId.
- TRDP_IP_ADDR_T destAddr
  IP address of destination for this publishing.
- UINT32 cycle
  Publishing cycle in us.
- UINT32 redId
  Redundancy group id.
- UINT32 redState
  Redundant state.Leader or Follower.
- UINT32 numPut
  Number of packet updates.
- UINT32 numSend
  Number of packets sent out.

4.23.1 Detailed Description

Table containing particular PD publishing information.

4.23.2 Field Documentation

4.23.2.1 TRDP_IP_ADDR_T TRDP_PUB_STATISTICS_T::destAddr

IP address of destination for this publishing.

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

- trdp_types.h

4.24 TRDP_RED_STATISTICS_T Struct Reference

A table containing PD redundant group information.

`#include <trdp_types.h>`

Data Fields

- UINT32 id

  Redundant Id.

- TRDP_RED_STATE_T state

  Redundant state.Leader or Follower.

4.24.1 Detailed Description

A table containing PD redundant group information.

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

- trdp_types.h

4.25 TRDP_SEND_PARAM_T Struct Reference

Quality/type of service and time to live.

`#include <trdp_types.h>`

4.25.1 Detailed Description

Quality/type of service and time to live.

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

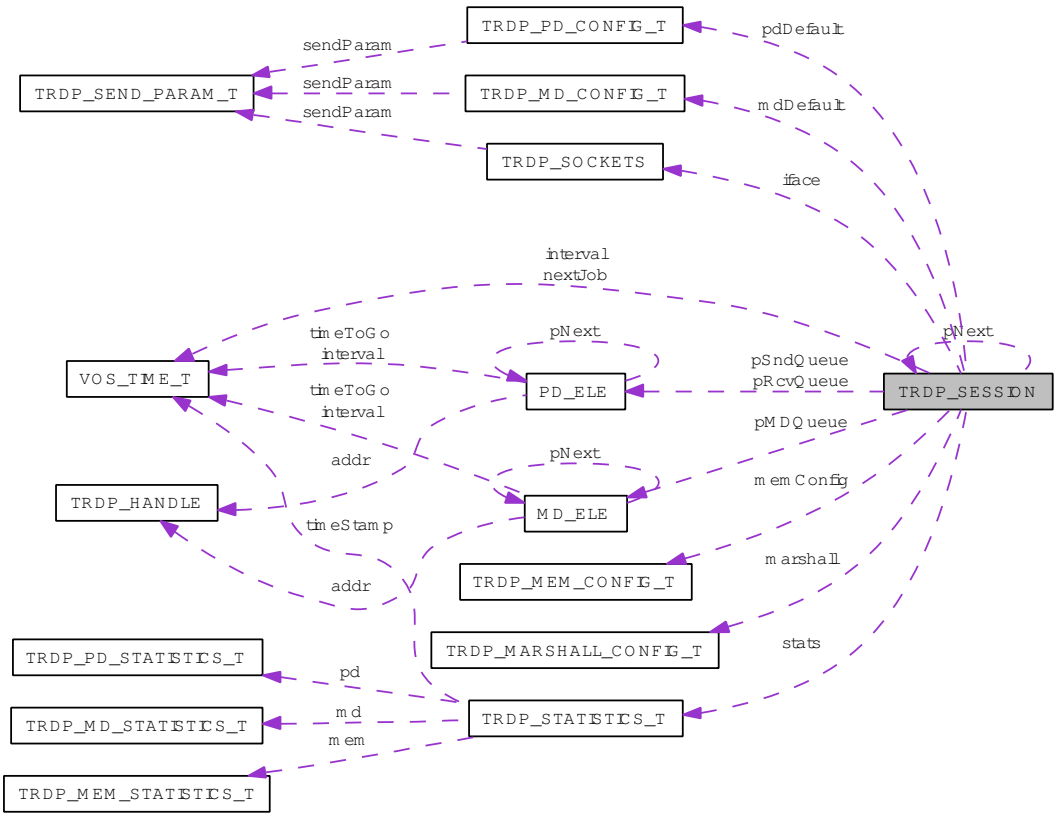
- trdp_types.h

4.26 TRDP_SESSION Struct Reference

Session/application variables store.

`#include <trdp_private.h>`

Collaboration diagram for TRDP_SESSION:



Data Fields

- struct TRDP_SESSION * pNext

  Pointer to next session.

- VOS_MUTEX_T mutex

  protect this session

- TRDP_IP_ADDR_T realIP

  Real IP address.

- TRDP_IP_ADDR_T virtualIP

  Virtual IP address.

- BOOL beQuiet

  if set, only react on ownIP requests

- UINT32 redID
  redundant comId
- UINT32 topoCount
  current valid topocount or zero
- TRDP_TIME_T interval
  Store for next select interval.
- TRDP_PD_CONFIG_T pdDefault
  Default configuration for process data.
- TRDP_SOCKETS_T iface [VOS_MAX_SOCKET_CNT]
  Collection of sockets to use.
- PD_ELE_T * pSndQueue
  pointer to first element of send queue
- PD_ELE_T * pRcvQueue
  pointer to first element of rcv queue
- MD_ELE_T * pMDQueue
  pointer to first element of MD session
- TRDP_STATISTICS_T stats
  statistics of this session

4.26.1 Detailed Description

Session/application variables store.

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

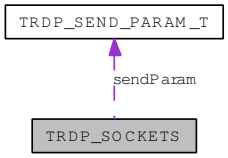
- trdp_private.h

4.27 TRDP_SOCKETS Struct Reference

Socket item.

`#include <trdp_private.h>`

Collaboration diagram for TRDP_SOCKETS:



Data Fields

- INT32 sock
  vos socket descriptor to use
- TRDP_IP_ADDR_T bindAddr
  Defines the interface to use.
- TRDP_SEND_PARAM_T sendParam
  Send parameters.
- TRDP_SOCK_TYPE_T type
  Usage of this socket.
- UINT16 usage
  No.

4.27.1 Detailed Description

Socket item.

4.27.2 Field Documentation

4.27.2.1 UINT16 TRDP_SOCKETS::usage

No.

of current users of this socket

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

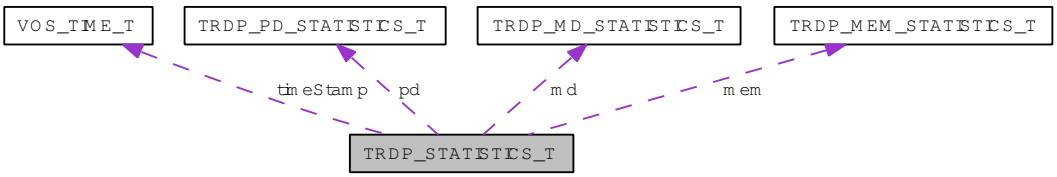
- trdp_private.h

4.28 TRDP_STATISTICS_T Struct Reference

Structure containing all general memory, PD and MD statistics information.

`#include <trdp_types.h>`

Collaboration diagram for TRDP_STATISTICS_T:



Data Fields

- UINT32 version

  TRDP version.

- TRDP_TIME_T timeStamp

  actual time stamp

- UINT32 upTime

  time in sec since last initialisation

- UINT32 statisticTime

  time in sec since last reset of statistics

- TRDP_LABEL_T hostName

  host name

- TRDP_LABEL_T leaderName

  leader host name

- TRDP_IP_ADDR_T ownIpAddr

  own IP address

- TRDP_IP_ADDR_T leaderIpAddr

  leader IP address

- UINT32 processPrio

  priority of TRDP process

- UINT32 processCycle

  cycle time of TRDP process in microseconds

- TRDP_MEM_STATISTICS_T mem

  memory statistics

- TRDP_PD_STATISTICS_T pd
  pd statistics
- TRDP_MD_STATISTICS_T md
  md statistics

4.28.1 Detailed Description

Structure containing all general memory, PD and MD statistics information.

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

- trdp_types.h

4.29 TRDP_SUBS_STATISTICS_T Struct Reference

Table containing particular PD subscription information.

`#include <trdp_types.h>`

Data Fields

- UINT32 comId

  Subscribed ComId.

- TRDP_IP_ADDR_T joinedAddr

  Joined IP address.

- TRDP_IP_ADDR_T filterAddr

  Filter IP address, i.e IP address of the sender for this subscription, 0.0.0.0 in case all senders.

- UINT32 callBack

  Reference for call back function if used.

- UINT32 timeout

  Time-out value in us.

- TRDP_ERR_T status

  Receive status information TRDP_NO_ERR, TRDP_TIMEOUT_ERR.

- TRDP_TO_BEHAVIOR_T toBehav

  Behaviour at time-out.

- UINT32 numRecv

  Number of packets received for this subscription.

4.29.1 Detailed Description

Table containing particular PD subscription information.

4.29.2 Field Documentation

4.29.2.1 TRDP_IP_ADDR_T TRDP_SUBS_STATISTICS_T::filterAddr

Filter IP address, i.e IP address of the sender for this subscription, 0.0.0.0 in case all senders.

4.29.2.2 UINT32 TRDP_SUBS_STATISTICS_T::timeout

Time-out value in us.

0 = No time-out supervision

4.29.2.3 TRDP_TO_BEHAVIOR_T TRDP_SUBS_STATISTICS_T::toBehav

Behaviour at time-out.

Set data to zero / keep last value

4.29.2.4 UINT32 TRDP_SUBS_STATISTICS_T::numRecv

Number of packets received for this subscription.

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

- trdp_types.h

4.30 TRDP_UIC_CAR_INFO_T Struct Reference

UIC car information structure.

`#include <tau_addr.h>`

Data Fields

- UINT8 cstProp [TRDP_UIC_CST_PROPERTY_LEN]
  consist properties
- UINT8 carProp [TRDP_UIC_CAR_PROPERTY_LEN]
  car properties
- UINT8 uicIdent [TRDP_UIC_IDENTIFIER_LEN]
  UIC identification number.
- UINT8 cstSeqNo
  consist sequence number in UIC Train
- UINT8 carSeqNo
  car sequence number in UIC ref direction
- UINT16 seatResNo
  car number for seat reservation
- INT8 contrCarCnt
  total number of controlled cars in consist
- UINT8 operat
  consist operator type (s.
- UINT8 owner
  consist owner type (s.
- UINT8 natAppl
  national application type (s.
- UINT8 natVer
  national application version (s.
- UINT8 trnOrient
  0 if car orientation is opposite to Train
- UINT8 cstOrient
  0 if car orientation is opposite to Consist
- UINT8 isLeading
  0 if car is not leading

- UINT8 isLeadRequ
  0 if no leading request
- UINT8 trnSwInCarCnt
  total number of train switches in car

4.30.1 Detailed Description

UIC car information structure.

4.30.2 Field Documentation

4.30.2.1 UINT8 TRDP_UIC_CAR_INFO_T::operat

consist operator type (s.
UIC 556)

4.30.2.2 UINT8 TRDP_UIC_CAR_INFO_T::owner

consist owner type (s.
UIC 556)

4.30.2.3 UINT8 TRDP_UIC_CAR_INFO_T::natAppl

national application type (s.
UIC 556)

4.30.2.4 UINT8 TRDP_UIC_CAR_INFO_T::natVer

national application version (s.
UIC 556)
The documentation for this struct was generated from the following file:

- tau_addr.h

4.31 TRDP_UIC_TRAIN_INFO_T Struct Reference

UIC train information structure.

`#include <tau_addr.h>`

Data Fields

- UINT32 trnCarCnt
  Total number of UIC cars.
- UINT8 confPos [TRDP_UIC_CONF_POS_LEN]
  confirmed position of unreachable cars
- UINT8 confPosAvail
  0 if conf.
- UINT8 operatAvail
  0 if operator/owner is not available
- UINT8 natApplAvail
  0 if national Application/Version is not available
- UINT8 cstPropAvail
  0 if UIC consist properties are not available
- UINT8 carPropAvail
  0 if UIC car properties are not available
- UINT8 seatResNoAvail
  0 if if reservation number is not available
- UINT8 inaugFrameVer
  inauguration frame version, s.
- UINT8 rDataVer
  supported R-Telegram Version, s.
- UINT8 inaugState
  UIC inaugaration state.
- UINT32 topoCnt
  UIC (i.e.
- UINT8 orient
  0 if UIC reference orientation is opposite to IPT
- UINT8 notAllConf
  0 if feature is not available

- UINT8 confCancelled
  0 if feature is not available

4.31.1 Detailed Description

UIC train information structure.

4.31.2 Field Documentation

4.31.2.1 UINT8 TRDP_UIC_TRAIN_INFO_T::confPosAvail

0 if conf.

Position is not available

4.31.2.2 UINT8 TRDP_UIC_TRAIN_INFO_T::inaugFrameVer

inauguration frame version, s.

Leaflet 556 Ann. C.3

4.31.2.3 UINT8 TRDP_UIC_TRAIN_INFO_T::rDataVer

supported R-Telegram Version, s.

Leaflet 556 Ann. A

4.31.2.4 UINT32 TRDP_UIC_TRAIN_INFO_T::topoCnt

UIC (i.e.

TCN) topography counter

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

- tau_addr.h

4.32 VOS_SOCK_OPT_T Struct Reference

Common socket options.

`#include <vos_sock.h>`

Data Fields

- UINT8 qos
  quality/type of service 0.
- UINT8 ttl
  time to live for unicast (default 64)
- UINT8 ttl_multicast
  time to live for multicast
- BOOL reuseAddrPort
  allow reuse of address and port
- BOOL nonBlocking
  use non blocking calls

4.32.1 Detailed Description

Common socket options.

4.32.2 Field Documentation

4.32.2.1 UINT8 VOS_SOCK_OPT_T::qos

quality/type of service 0.

..7

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

- vos_sock.h

4.33 VOS_TIME_T Struct Reference

Timer value compatible with timeval / select.

`#include <vos_types.h>`

Data Fields

- UINT32 tv_sec
  full seconds
- UINT32 tv_usec
  Micro seconds (max.

4.33.1 Detailed Description

Timer value compatible with timeval / select.

Relative or absolute date, depending on usage

4.33.2 Field Documentation

4.33.2.1 UINT32 VOS_TIME_T::tv_usec

Micro seconds (max.

value 999999)

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

- vos_types.h

Chapter 5

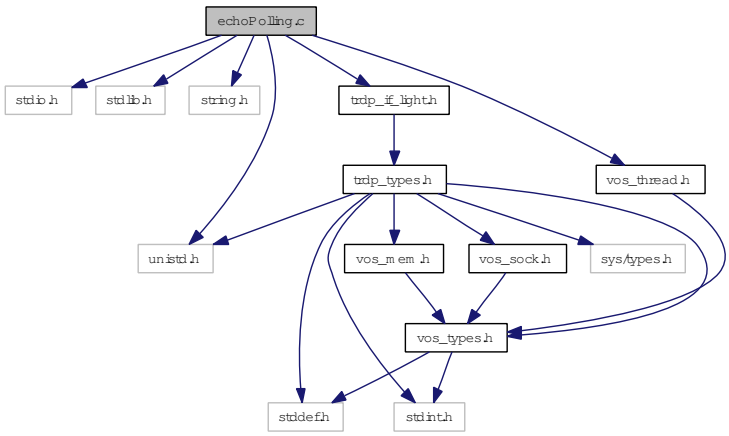
File Documentation

5.1 echoPolling.c File Reference

Demo echoing application for TRDP.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "trdp_if_light.h"
#include "vos_thread.h"
```

Include dependency graph for echoPolling.c:



Functions

- void dbgOut (void *pRefCon, TRDP_LOG_T category, const CHAR8 *pTime, const CHAR8 *pFile, UINT16 LineNumber, const CHAR8 *pMsgStr)

  callback routine for TRDP logging/error output

- int main (int argc, char **argv)

  main entry

5.1.1 Detailed Description

Demo echoing application for TRDP.

Receive and send process data, single threaded polling, static memory

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

echoPolling.c 2 2012-06-04 11:25:16Z 97025

5.1.2 Function Documentation

5.1.2.1 void dbgOut (void * *pRefCon*, TRDP_LOG_T *category*, const CHAR8 * *pTime*, const CHAR8 * *pFile*, UINT16 *LineNumber*, const CHAR8 * *pMsgStr*)

callback routine for TRDP logging/error output

Parameters:

← ***pRefCon*** user supplied context pointer

← ***category*** Log category (Error, Warning, Info etc.)

← ***pTime*** pointer to NULL-terminated string of time stamp

← ***pFile*** pointer to NULL-terminated string of source module

← ***LineNumber*** line

← ***pMsgStr*** pointer to NULL-terminated string

Return values:

none

5.1.2.2 int main (int *argc*, char ** *argv*)

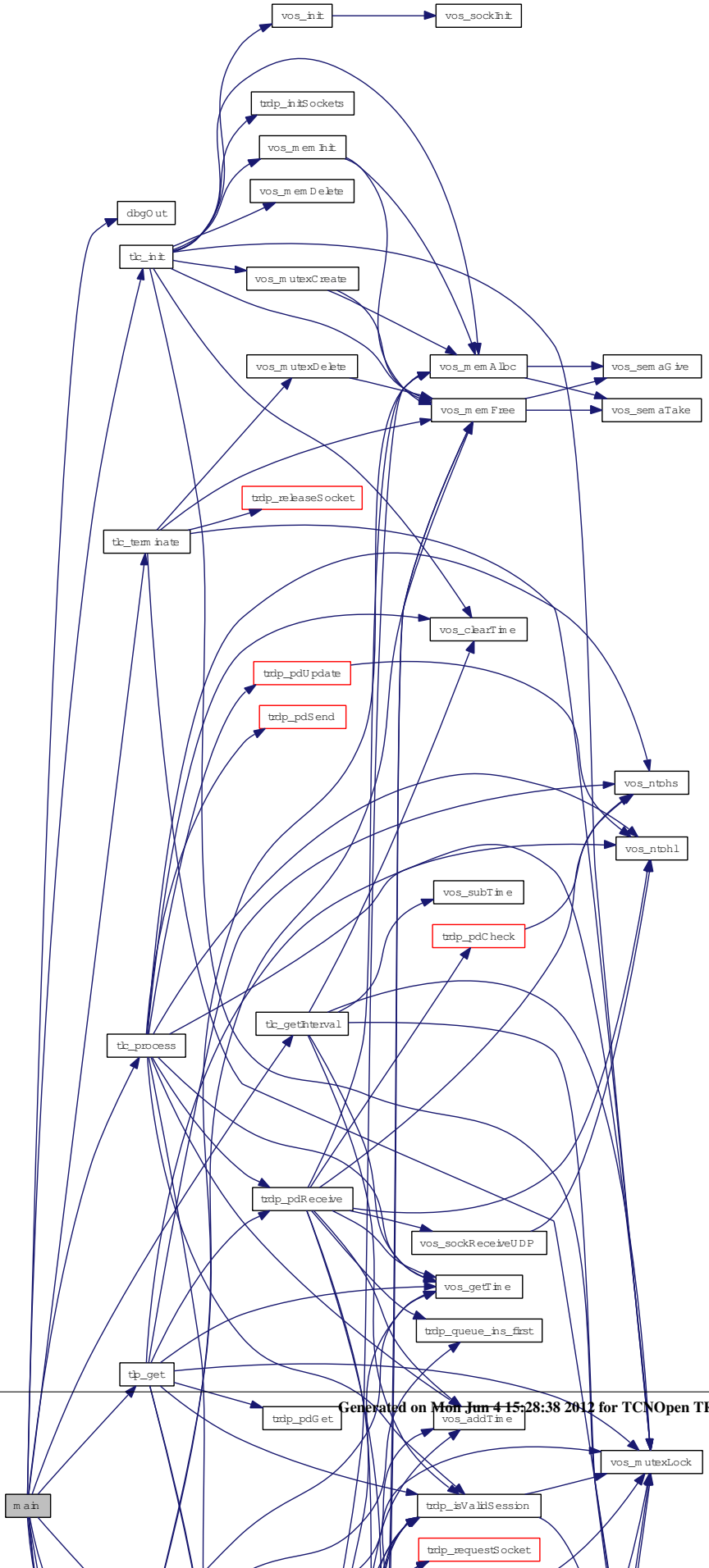
main entry

Return values:

0 no error

1 some error

Here is the call graph for this function:

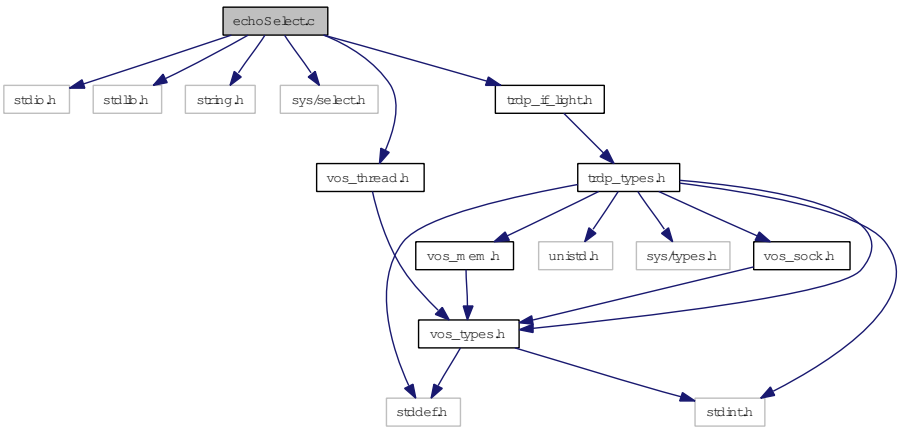


5.2 echoSelect.c File Reference

Demo echoing application for TRDP.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/select.h>
#include "trdp_if_light.h"
#include "vos_thread.h"
```

Include dependency graph for echoSelect.c:



Functions

- void dbgOut (void *pRefCon, TRDP_LOG_T category, const CHAR8 *pTime, const CHAR8 *pFile, UINT16 LineNumber, const CHAR8 *pMsgStr)

  callback routine for TRDP logging/error output

- void myPDcallBack (void *pRefCon, const TRDP_PD_INFO_T *pMsg, UINT8 *pData, UINT32 dataSize)

  callback routine for receiving TRDP traffic

- int main (int argc, char **argv)

  main entry

5.2.1 Detailed Description

Demo echoing application for TRDP.

Receive and send process data, single threaded using select() and heap memory

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

echoSelect.c 2 2012-06-04 11:25:16Z 97025

5.2.2 Function Documentation

5.2.2.1 void dbgOut (void * *pRefCon*, TRDP_LOG_T *category*, const CHAR8 * *pTime*, const CHAR8 * *pFile*, UINT16 *LineNumber*, const CHAR8 * *pMsgStr*)

callback routine for TRDP logging/error output

Parameters:

← ***pRefCon*** user supplied context pointer

← ***category*** Log category (Error, Warning, Info etc.)

← ***pTime*** pointer to NULL-terminated string of time stamp

← ***pFile*** pointer to NULL-terminated string of source module

← ***LineNumber*** line

← ***pMsgStr*** pointer to NULL-terminated string

Return values:

none

5.2.2.2 int main (int *argc*, char ** *argv*)

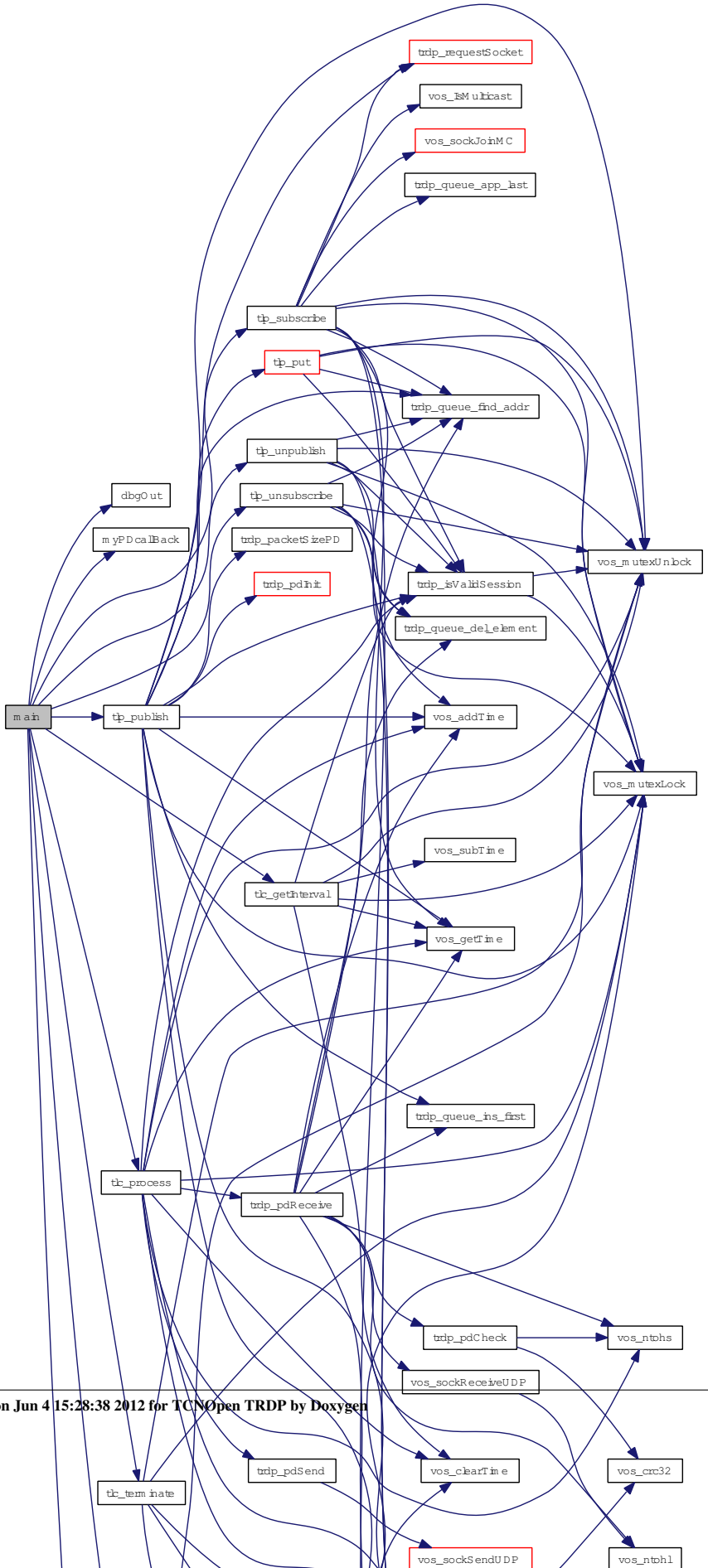
main entry

Return values:

0 no error

1 some error

Here is the call graph for this function:



5.2.2.3 void myPDcallBack (void * *pRefCon*, const TRDP_PD_INFO_T * *pMsg*, UINT8 * *pData*, UINT32 *dataSize*)

callback routine for receiving TRDP traffic

Parameters:

← ***pRefCon*** user supplied context pointer

← ***pMsg*** pointer to header/packet infos

← ***pData*** pointer to data block

← ***dataSize*** pointer to data size

Return values:

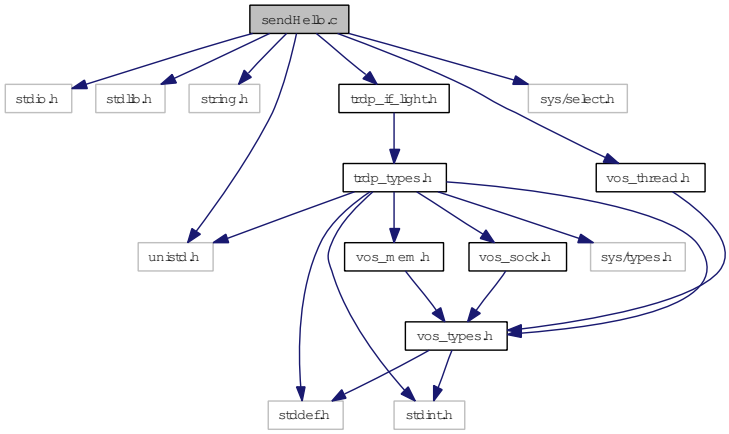
none

5.3 sendHello.c File Reference

Demo application for TRDP.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/select.h>
#include "trdp_if_light.h"
#include "vos_thread.h"
```

Include dependency graph for sendHello.c:



Functions

- int main (int argc, char *argv[ ])
  main entry

5.3.1 Detailed Description

Demo application for TRDP.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr and Florian Weispfenning, NewTec GmbH

Remarks:

Id

sendHello.c 2 2012-06-04 11:25:16Z 97025

5.3.2 Function Documentation

5.3.2.1 int main (int *argc*, char * *argv*[ ])

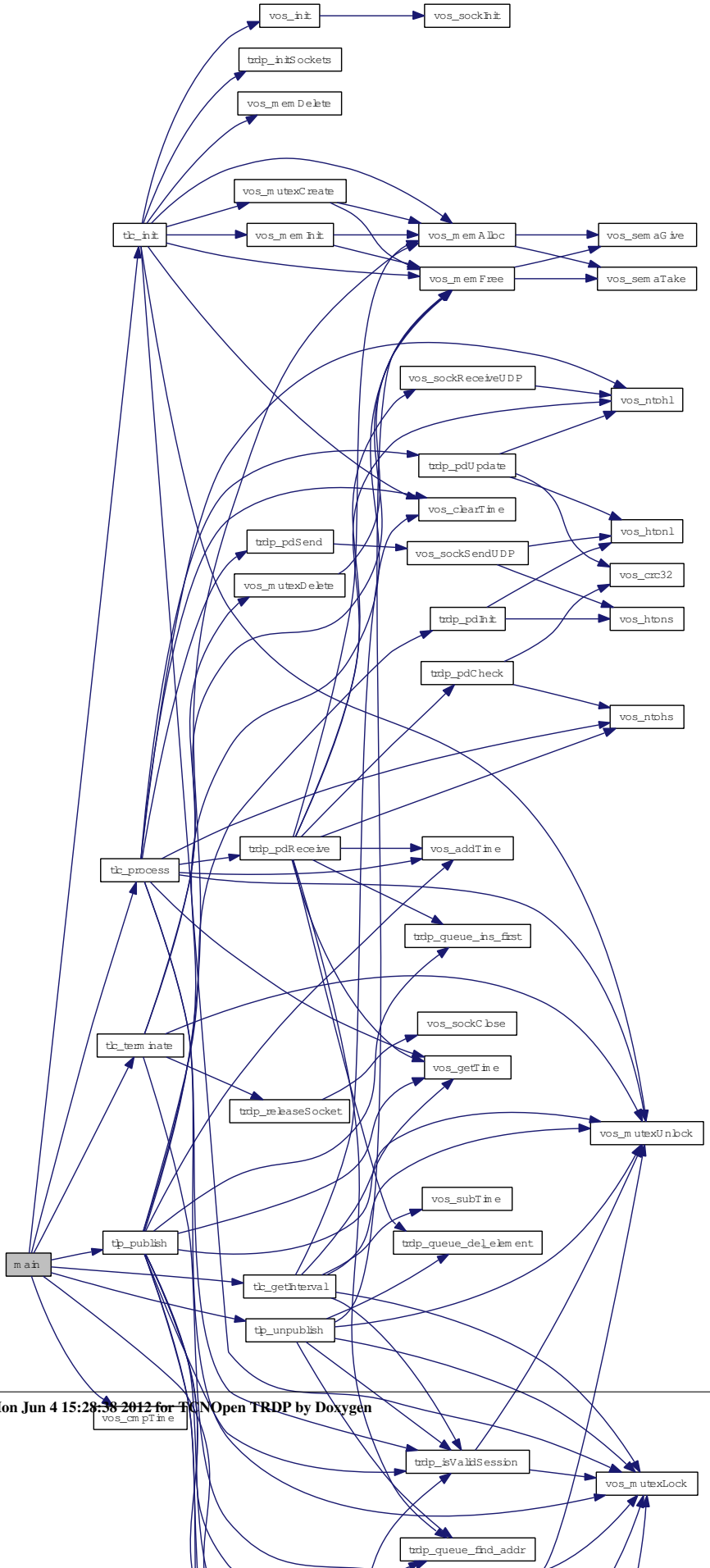
main entry

Return values:

0 no error

1 some error

Here is the call graph for this function:

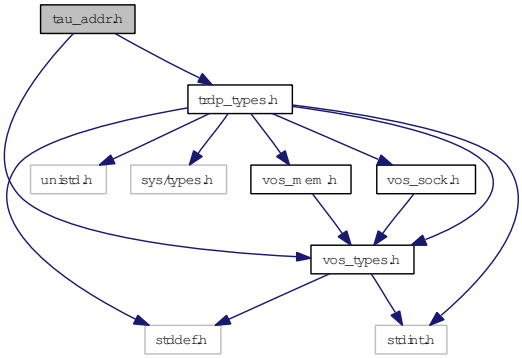


5.4 tau_addr.h File Reference

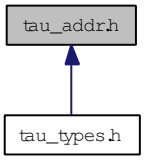
TRDP utility interface definitions.

```
#include "vos_types.h"
#include "trdp_types.h"
```

Include dependency graph for tau_addr.h:



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



Data Structures

- struct TRDP_DEVICE_INFO_T
  device information structure
- struct TRDP_CAR_INFO_T
  car information structure.
- struct TRDP_UIC_TRAIN_INFO_T
  UIC train information structure.
- struct TRDP_UIC_CAR_INFO_T
  UIC car information structure.

Enumerations

- enum TRDP_INAUGSTATE_T {
  TRDP_INAUGSTATE_FAULT,
  TRDP_INAUGSTATE_OK = 2 }

Types for address conversion and inauguration information.

Functions

- EXT_DECL TRDP_ERR_T tau_getTrnBackboneType (UINT8 *pTbType, TRDP_IP_ADDR *pGatewayIpAddr)

  Function to retrieve the train backbone type.

- EXT_DECL TRDP_ERR_T tau_getEtbState (TRDP_INAUGSTATE_T *pInaugState, UINT32 *pTopoCnt)

  Function to retrieve the inauguration state and the topography counter.

- EXT_DECL TRDP_ERR_T tau_getOwnIds (TRDP_LABEL_T devId, TRDP_LABEL_T carId, TRDP_LABEL_T cstId)

  Who am I ?.

- EXT_DECL TRDP_ERR_T tau_getAddrByName (const TRDP_URI_T uri, TRDP_IP_ADDR *pIpAddr, UINT32 *pTopoCnt)

  Function to convert a URI to an IP address.

- EXT_DECL TRDP_ERR_T tau_getUriHostPart (TRDP_IP_ADDR ipAddr, TRDP_URI_HOST_T uri, UINT32 *pTopoCnt)

  Function to get the host part of an URI.

- EXT_DECL TRDP_ERR_T tau_label2CarId (const TRDP_LABEL_T carLabel, const TRDP_LABEL_T cstLabel, TRDP_LABEL_T pCarId, UINT32 *pTopoCnt)

  Function to convert a label to a carID.

- EXT_DECL TRDP_ERR_T tau_label2CarNo (const TRDP_LABEL_T carLabel, const TRDP_LABEL_T cstLabel, UINT8 *pCarNo, UINT32 *pTopoCnt)

  The function delivers the car number to the given label.

- EXT_DECL TRDP_ERR_T tau_addr2CarId (TRDP_IP_ADDR ipAddr, TRDP_LABEL_T carId, UINT32 *pTopoCnt)

  Function to get the carId from an IP address.

- EXT_DECL TRDP_ERR_T tau_label2CstId (const TRDP_LABEL_T carLabel, TRDP_LABEL_T cstId, UINT32 *pTopoCnt)

  Function to.

- EXT_DECL TRDP_ERR_T tau_addr2CstId (TRDP_IP_ADDR ipAddr, TRDP_LABEL_T cstId, UINT32 *pTopoCnt)

  Function to.

- EXT_DECL TRDP_ERR_T tau_cstNo2CstId (UINT8 trnCstNo, TRDP_LABEL_T cstId, UINT32 *pTopoCnt)

  Function to.

- EXT_DECL TRDP_ERR_T tau_label2TrnCstNo (const TRDP_LABEL_T carLabel, UINT8 *pTrnCstNo, UINT32 *pTopoCnt)

Function to.

- EXT_DECL TRDP_ERR_T tau_addr2TrnCstNo (TRDP_IP_ADDR ipAddr, UINT8 *pTrnCstNo, UINT32 *pTopoCnt)

  Function to.

- EXT_DECL TRDP_ERR_T tau_getTrnCstCnt (UINT8 *pCstCnt, UINT32 *pTopoCnt)

  Function to.

- EXT_DECL TRDP_ERR_T tau_getCstCarCnt (const TRDP_LABEL_T cstLabel, UINT8 *pCarCnt, UINT32 *pTopoCnt)

  Function to.

- EXT_DECL TRDP_ERR_T tau_getCarDevCnt (const TRDP_LABEL_T carLabel, const TRDP_LABEL_T cstLabel, UINT16 *pDevCnt, UINT32 *pTopoCnt)

  Function to.

- EXT_DECL TRDP_ERR_T tau_getCarInfo (const TRDP_LABEL_T carLabel, const TRDP_LABEL_T cstLabel, UINT16 maxDev, TRDP_CAR_INFO_T *pCarData, UINT32 *pTopoCnt)

  Function to tbd.

- EXT_DECL TRDP_ERR_T tau_getUicState (UINT8 **pInaugState, UINT32 **pTopoCnt)

  Function to tbd.

- EXT_DECL TRDP_ERR_T tau_getUicCarData (UINT8 carSeqNo, TRDP_UIC_CAR_DATA_T *pCarData, UINT32 *pTopoCnt)

  Function to tbd.

- EXT_DECL TRDP_ERR_T tau_Label2UicCarSeqNo (const TRDP_LABEL_T carLabel, const TRDP_LABEL_T cstLabel, UINT8 *pCarSeqNo, UINT32 *pTopoCnt)

  Function to tbd.

- EXT_DECL TRDP_ERR_T tau_Addr2UicCarSeqNo (TRDP_IP_ADDR ipAddr, UINT8 *pCarSeqNo, UINT8 *pTopoCnt)

  Function to tbd.

- EXT_DECL TRDP_ERR_T tau_UicCarSeqNo2Ids (UINT8 carSeqNo, TRDP_LABEL_T carId, TRDP_LABEL_T cstId, UINT32 *pTopoCnt)

  Function to tbd.

- EXT_DECL TRDP_ERR_T tau_getCarOrient (TRDP_LABEL_T carId, TRDP_LABEL_T cstId, UINT8 *pCarOrient, UINT8 *pCstOrient, UINT8 *pUicCarOrient, UINT8 *pUicCstOrient, UINT32 *pTopoCnt)

  Function to retrieve the orientation of the given car.

- EXT_DECL TRDP_ERR_T tau_getCarOrient (TRDP_LABEL_T cstId, UINT8 *pCstOrient, UINT8 *pUicCstOrient, UINT32 *pTopoCnt)

  Function to retrieve the orientation of the given consist.

5.4.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- IP - URI address translation and train configuration access

Note:

Project: TCNOpen TRDP prototype stack

Author:

Armin-H. Weiss (initial version)

Remarks:

Id

tau_addr.h 2 2012-06-04 11:25:16Z 97025

5.4.2 Enumeration Type Documentation

5.4.2.1 enum TRDP_INAUGSTATE_T

Types for address conversion and inauguration information.

inauguration states

Enumerator:

TRDP_INAUGSTATE_FAULT DNS not or not yet available, no address transformation possible. ongoing train inauguration,trainwide communication not possible

TRDP_INAUGSTATE_OK inauguration done, trainwide communication possible

5.4.3 Function Documentation

5.4.3.1 EXT_DECL TRDP_ERR_T tau_addr2CarId (TRDP_IP_ADDR *ipAddr*, TRDP_LABEL_T *carId*, UINT32 * *pTopoCnt*)

Function to get the carId from an IP address.

Parameters:

← ***ipAddr*** IP address

→ ***carId*** Car ID

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.2 EXT_DECL TRDP_ERR_T tau_addr2CstId (TRDP_IP_ADDR *ipAddr*, TRDP_LABEL_T *cstId*, UINT32 * *pTopoCnt*)

Function to.

Parameters:

← ***ipAddr***

→ ***cstId***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.3 EXT_DECL TRDP_ERR_T tau_addr2TrnCstNo (TRDP_IP_ADDR *ipAddr*, UINT8 * *pTrnCstNo*, UINT32 * *pTopoCnt*)

Function to.

Parameters:

← ***ipAddr*** IP address

→ ***pTrnCstNo***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.4 EXT_DECL TRDP_ERR_T tau_Addr2UicCarSeqNo (TRDP_IP_ADDR *ipAddr*, UINT8 * *pCarSeqNo*, UINT8 * *pTopoCnt*)

Function to tbd.

Parameters:

← ***ipAddr*** tbd

→ ***pCarSeqNo*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.5 EXT_DECL TRDP_ERR_T tau_cstNo2CstId (UINT8 *trnCstNo*, TRDP_LABEL_T *cstId*, UINT32 * *pTopoCnt*)

Function to.

Parameters:

← ***trnCstNo***

→ ***cstId***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.6 EXT_DECL TRDP_ERR_T tau_getAddrByName (const TRDP_URI_T *uri*, TRDP_IP_ADDR * *pIpAddr*, UINT32 * *pTopoCnt*)

Function to convert a URI to an IP address.

Receives a URI as input variable and translates this URI to an IP-Address. The URI may specify either a unicast or a multicast IP-Address. The caller may specify a topographic counter, which will be checked.

Parameters:

← ***uri*** Pointer to a URI or an IP Address string

→ ***pIpAddr*** Pointer to return the IP address

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.7 EXT_DECL TRDP_ERR_T tau_getCarDevCnt (const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*, UINT16 * *pDevCnt*, UINT32 * *pTopoCnt*)

Function to.

Parameters:

← ***carLabel***

← ***cstLabel***

→ ***pDevCnt***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.8 EXT_DECL TRDP_ERR_T tau_getCarInfo (const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*, UINT16 *maxDev*, TRDP_CAR_INFO_T * *pCarData*, UINT32 * *pTopoCnt*)

Function to tbd.

Parameters:

← ***carLabel*** tbd

← ***cstLabel*** tbd

← ***maxDev*** tbd

→ ***pCarData*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.9 **********EXT_DECL TRDP_ERR_T tau_getCarOrient (TRDP_LABEL_T *cstId*, UINT8 * *pCstOrient*, UINT8 * *pUicCstOrient*, UINT32 * *pTopoCnt*)

Function to retrieve the orientation of the given consist.

Parameters:

← ***cstId*** cstId = NULL means own consist

→ ***pCstOrient*** tbd

→ ***pUicCstOrient*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

Parameters:

← ***cstId*** cstId = NULL means own consist

→ ***pCstOrient*** tbd

→ ***pUicCstOrient*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.10 EXT_DECL TRDP_ERR_T tau_getCarOrient (TRDP_LABEL_T *carId*, TRDP_LABEL_T *cstId*, UINT8 * *pCarOrient*, UINT8 * *pCstOrient*, UINT8 * *pUicCarOrient*, UINT8 * *pUicCstOrient*, UINT32 * *pTopoCnt*)

Function to retrieve the orientation of the given car.

Parameters:

← ***carId*** carId = NULL means own car

← ***cstId*** cstId = NULL means own consist

→ ***pCarOrient*** tbd

→ ***pCstOrient*** tbd

→ ***pUicCarOrient*** tbd

→ ***pUicCstOrient*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.11 EXT_DECL TRDP_ERR_T tau_getCstCarCnt (const TRDP_LABEL_T *cstLabel*, UINT8 * *pCarCnt*, UINT32 * *pTopoCnt*)

Function to.

Parameters:

← ***cstLabel***

→ ***pCarCnt***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.12 EXT_DECL TRDP_ERR_T tau_getEtbState (TRDP_INAUGSTATE_T * *pInaugState*, UINT32 * *pTopoCnt*)

Function to retrieve the inauguration state and the topography counter.

Parameters:

→ ***pInaugState*** Pointer to an inauguration state variable.

→ ***pTopoCnt*** Pointer to a topo counter variable.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.13 EXT_DECL TRDP_ERR_T tau_getOwnIds (TRDP_LABEL_T *devId*, TRDP_LABEL_T *carId*, TRDP_LABEL_T *cstId*)

Who am I ?.

Realizes a kind of Who am I function. It is used to determine the own identifiers (i.e. the own labels), which may be used as host part of the own fully qualified domain name.

Parameters:

→ ***devId*** Returns the device label (host name)

→ ***carId*** Returns the car label

→ ***cstId*** Returns the consist label

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.14 EXT_DECL TRDP_ERR_T tau_getTrnBackboneType (UINT8 * *pTbType*, TRDP_IP_ADDR * *pGatewayIpAddr*)

Function to retrieve the train backbone type.

Parameters:

→ ***pTbType*** Pointer to return the train backbone type. 0=ETB, 1= WTB

→ ***pGatewayIpAddr*** IP address of active gateway to train backbone. This parameter may be a NULL pointer if the caller is not interested in the address.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.15 EXT_DECL TRDP_ERR_T tau_getTrnCstCnt (UINT8 * *pCstCnt*, UINT32 * *pTopoCnt*)

Function to.

Parameters:

→ ***pCstCnt***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.16 EXT_DECL TRDP_ERR_T tau_getUicCarData (UINT8 *carSeqNo*, TRDP_UIC_CAR_DATA_T * *pCarData*, UINT32 * *pTopoCnt*)

Function to tbd.

Parameters:

← ***carSeqNo*** tbd

→ ***pCarData*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.17 EXT_DECL TRDP_ERR_T tau_getUicState (UINT8 ** *pInaugState*, UINT32 ** *pTopoCnt*)

Function to tbd.

Parameters:

→ ***pInaugState*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.18 EXT_DECL TRDP_ERR_T tau_getUriHostPart (TRDP_IP_ADDR *ipAddr*, TRDP_URI_HOST_T *uri*, UINT32 * *pTopoCnt*)

Function to get the host part of an URI.

Receives an IP-Address and translates it into the host part of the corresponding URI; both unicast and multicast addresses are accepted. The caller may specify a topographic counter, which will be checked.

Parameters:

← ***ipAddr*** IP address

→ ***uri***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.19 EXT_DECL TRDP_ERR_T tau_label2CarId (const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*, TRDP_LABEL_T *pCarId*, UINT32 * *pTopoCnt*)

Function to convert a label to a carID.

Parameters:

← ***carLabel*** Car label

← ***cstLabel*** Consist label

→ ***pCarId*** Pointer to the carID returned

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.20 EXT_DECL TRDP_ERR_T tau_label2CarNo (const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*, UINT8 * *pCarNo*, UINT32 * *pTopoCnt*)

The function delivers the car number to the given label.

The first match of the table will be returned in case there is no unique label given.

Parameters:

← ***carLabel*** Car label

← ***cstLabel*** Consist label

→ ***pCarNo*** Pointer to the carNo returned

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.21 EXT_DECL TRDP_ERR_T tau_label2CstId (const TRDP_LABEL_T *carLabel*, TRDP_LABEL_T *cstId*, UINT32 * *pTopoCnt*)

Function to.

Parameters:

← ***carLabel***

→ ***cstId***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.22 EXT_DECL TRDP_ERR_T tau_label2TrnCstNo (const TRDP_LABEL_T *carLabel*, UINT8 * *pTrnCstNo*, UINT32 * *pTopoCnt*)

Function to.

Parameters:

← ***carLabel***

→ ***pTrnCstNo***

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.23 EXT_DECL TRDP_ERR_T tau_Label2UicCarSeqNo (const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*, UINT8 * *pCarSeqNo*, UINT32 * *pTopoCnt*)

Function to tbd.

Parameters:

← ***carLabel*** tbd

← ***cstLabel*** tbd

→ ***pCarSeqNo*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR Parameter error

5.4.3.24 EXT_DECL TRDP_ERR_T tau_UicCarSeqNo2Ids (UINT8 *carSeqNo*, TRDP_LABEL_T *carId*, TRDP_LABEL_T *cstId*, UINT32 * *pTopoCnt*)

Function to tbd.

Parameters:

← ***carSeqNo*** tbd

→ ***carId*** tbd

→ ***cstId*** tbd

↔ ***pTopoCnt*** Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

TRDP_NO_ERR no error

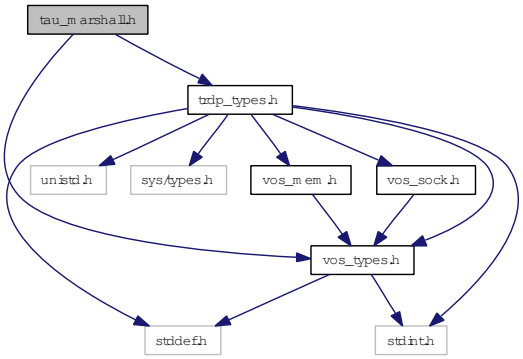
TRDP_PARAM_ERR Parameter error

5.5 tau_marshall.h File Reference

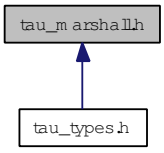
TRDP utility interface definitions.

```
#include "vos_types.h"
#include "trdp_types.h"
```

Include dependency graph for tau_marshall.h:



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



Typedefs

- typedef TRDP_ERR_T tau_marshall (void *pRefCon, UINT32 comId, const UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)

  marshall function.

- typedef TRDP_ERR_T tau_unmarshall (void *pRefCon, UINT32 comId, const UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)

  unmarshall function.

Functions

- EXT_DECL TRDP_ERR_T tau_initMarshall (void **ppRefCon, UINT32 numDataSet, TRDP_DATASET_T *pDataset)

  Types for marshalling / unmarshalling.

5.5.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- marshalling/unmarshalling

Note:

Project: TCNOpen TRDP prototype stack

Author:

Armin-H. Weiss (initial version)

Remarks:

Id

tau_marshall.h 2 2012-06-04 11:25:16Z 97025

5.5.2 Typedef Documentation

5.5.2.1 typedef TRDP_ERR_T tau_marshall(void *pRefCon, UINT32 comId, const UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)

marshall function.

Parameters:

← ***pRefCon*** pointer to user context

← ***comId*** ComId to identify the structure out of a configuration

← ***pSrc*** pointer to received original message

← ***pDest*** pointer to a buffer for the treated message

↔ ***pDestSize*** size of the provide buffer / size of the treated message

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR provided buffer to small

TRDP_INIT_ERR marshalling not initialised

TRDP_COMID_ERR comid not existing

5.5.2.2 typedef TRDP_ERR_T tau_unmarshall(void *pRefCon, UINT32 comId, const UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)

unmarshall function.

Parameters:

← ***pRefCon*** pointer to user context

← ***comId*** ComId to identify the structure out of a configuration

← ***pSrc*** pointer to received original message

← ***pDest*** pointer to a buffer for the treated message

↔ ***pDestSize*** size of the provide buffer / size of the treated message

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR provided buffer to small

TRDP_INIT_ERR marshalling not initialised

TRDP_COMID_ERR comid not existing

5.5.3 Function Documentation

5.5.3.1 EXT_DECL TRDP_ERR_T tau_initMarshall (void ** *ppRefCon*, UINT32 *numDataSet*, TRDP_DATASET_T * *pDataset*)

Types for marshalling / unmarshalling.

Function to initialise the marshalling/unmarshalling.

Parameters:

↔ ***ppRefCon*** Returns a pointer to be used for the reference context of marshalling/unmarshalling

← ***numDataSet*** Number of datasets found in the configuration

← ***pDataset*** Pointer to an array of a structures of type TRDP_DATASET_T

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR provided buffer to small

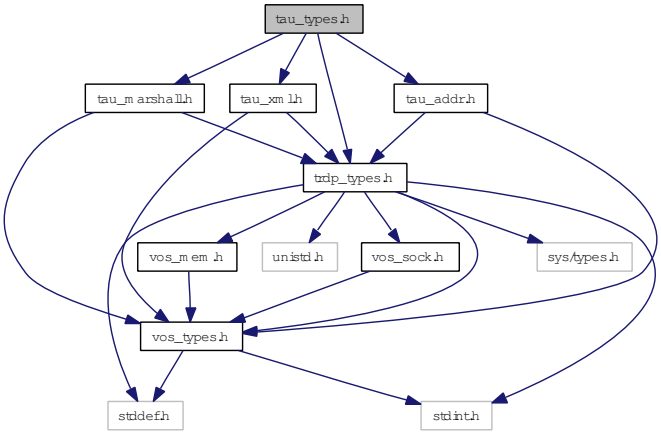
TRDP_PARAM_ERR Parameter error

5.6 tau_types.h File Reference

TRDP utility interface definitions.

```
#include "trdp_types.h"
#include "tau_addr.h"
#include "tau_marshall.h"
#include "tau_xml.h"
```

Include dependency graph for tau_types.h:



5.6.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- marshalling/unmarshalling
- xml configuration interpreter
- IP - URI address translation

Note:

Project: TCNOpen TRDP prototype stack

Author:

Armin-H. Weiss (initial version)

Remarks:

Id

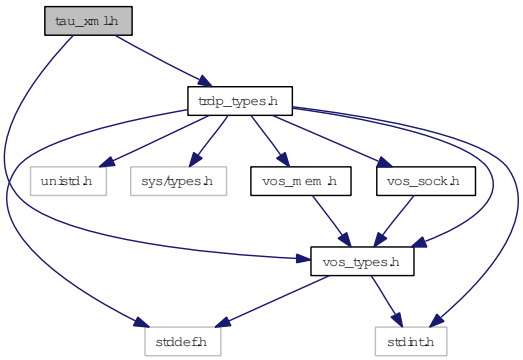
tau_types.h 2 2012-06-04 11:25:16Z 97025

5.7 tau_xml.h File Reference

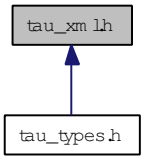
TRDP utility interface definitions.

```
#include "vos_types.h"
#include "trdp_types.h"
```

Include dependency graph for tau_xml.h:



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



Data Structures

- struct TRDP_PROCESS_CONFIG_T

  Types to read out the XML configuration.

- struct TRDP_DBG_CONFIG_T

  Control for debug output device/file on application level.

Enumerations

- enum TRDP_DBG_OPTION_T {
  TRDP_DBG_DEFAULT = 0,
  TRDP_DBG_OFF = 0x01,
  TRDP_DBG_ERR = 0x02,
  TRDP_DBG_WARN = 0x04,
  TRDP_DBG_INFO = 0x08,
  TRDP_DBG_DBG = 0x10,

TRDP_DBG_TIME = 0x20,
TRDP_DBG_LOC = 0x40,
TRDP_DBG_CAT = 0x80 }

Control for debug output format on application level.

Functions

- EXT_DECL TRDP_ERR_T tau_readXmlConfig (const CHAR8 *pFileName, TRDP_PROCESS_CONFIG_T *pProcessConfig, TRDP_MEM_CONFIG_T *pMemConfig, TRDP_PD_CONFIG_T *pPdConfig, TRDP_MD_CONFIG_T *pMdConfig, UINT32 *pNumExchgPar, TRDP_EXCHG_PAR_T **ppExchgPar, UINT32 *pNumComPar, TRDP_COM_PAR_T **ppComPar, TRDP_DBG_CONFIG_T *pDbgPar)

  Function to read the TRDP configuration parameters out of the XML configuration file.

- EXT_DECL TRDP_ERR_T tau_readXmlDatasetConfig (const CHAR8 *pFileName, UINT32 *pNumDataset, TRDP_DATASET_T **ppDataset)

  Function to read the DataSet configuration out of the XML configuration file.

5.7.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- read xml configuration interpreter

Note:

Project: TCNOpen TRDP prototype stack

Author:

Armin-H. Weiss (initial version)

Remarks:

Id

tau_xml.h 2 2012-06-04 11:25:16Z 97025

5.7.2 Enumeration Type Documentation

5.7.2.1 enum TRDP_DBG_OPTION_T

Control for debug output format on application level.

Enumerator:

TRDP_DBG_DEFAULT Printout default.

TRDP_DBG_OFF Printout off.

TRDP_DBG_ERR Printout error.

TRDP_DBG_WARN Printout warning and error.

TRDP_DBG_INFO Printout info, warning and error.

TRDP_DBG_DBG Printout debug, info, warning and error.

TRDP_DBG_TIME Printout timestamp.

TRDP_DBG_LOC Printout file name and line.

TRDP_DBG_CAT Printout category (DBG, INFO, WARN, ERR).

5.7.3 Function Documentation

5.7.3.1 EXT_DECL TRDP_ERR_T tau_readXmlConfig (const CHAR8 * *pFileName*, TRDP_PROCESS_CONFIG_T * *pProcessConfig*, TRDP_MEM_CONFIG_T * *pMemConfig*, TRDP_PD_CONFIG_T * *pPdConfig*, TRDP_MD_CONFIG_T * *pMdConfig*, UINT32 * *pNumExchgPar*, TRDP_EXCHG_PAR_T ** *ppExchgPar*, UINT32 * *pNumComPar*, TRDP_COM_PAR_T ** *ppComPar*, TRDP_DBG_CONFIG_T * *pDbgPar*)

Function to read the TRDP configuration parameters out of the XML configuration file.

Parameters:

← ***pFileName*** Path and filename of the xml configuration file

→ ***pProcessConfig*** TRDP main process configuration

→ ***pMemConfig*** Memory configuration

→ ***pPdConfig*** PD default configuration

→ ***pMdConfig*** MD default configuration

→ ***pNumExchgPar*** Number of configured telegrams

→ ***ppExchgPar*** Pointer to array of telegram configurations

→ ***pNumComPar*** Number of configured com parameters

→ ***ppComPar*** Pointer to array of com parameters

→ ***pDbgPar*** Debug printout options for application use

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR provided buffer to small

TRDP_PARAM_ERR File not existing

5.7.3.2 EXT_DECL TRDP_ERR_T tau_readXmlDatasetConfig (const CHAR8 * *pFileName*, UINT32 * *pNumDataset*, TRDP_DATASET_T ** *ppDataset*)

Function to read the DataSet configuration out of the XML configuration file.

Parameters:

← ***pFileName*** Path and filename of the xml configuration file

→ ***pNumDataset*** Pointer to the number of datasets found in the configuration

→ ***ppDataset*** Pointer to an array of a structures of type TRDP_DATASET_T

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR provided buffer to small

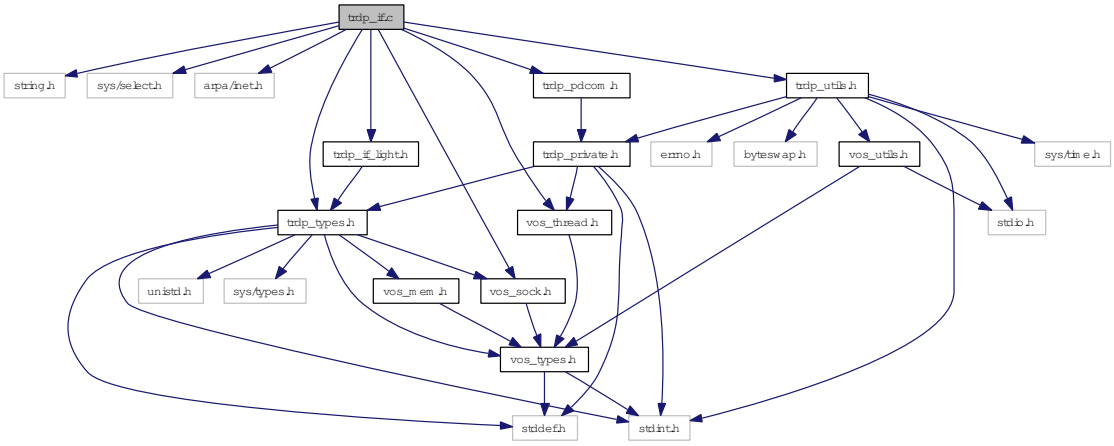
TRDP_PARAM_ERR File not existing

5.8 trdp_if.c File Reference

Functions for ECN communication.

```
#include <string.h>
#include <sys/select.h>
#include <arpa/inet.h>
#include "trdp_types.h"
#include "trdp_if_light.h"
#include "trdp_utils.h"
#include "trdp_pdcom.h"
#include "vos_thread.h"
#include "vos_sock.h"
```

Include dependency graph for trdp_if.c:



Functions

- BOOL trdp_isValidSession (TRDP_APP_SESSION_T pSessionHandle)

  Check if the session handle is valid.

- TRDP_APP_SESSION_T * trdp_sessionQueue (void)

  Get the session queue head pointer.

- EXT_DECL TRDP_ERR_T tlc_init (TRDP_APP_SESSION_T *pAppHandle, TRDP_IP_ADDR_T ownIpAddr, TRDP_IP_ADDR_T leaderIpAddr, const TRDP_PRINT_DBG_T pPrintDebugString, const TRDP_MARSHALL_CONFIG_T *pMarshall, const TRDP_PD_CONFIG_T *pPdDefault, const TRDP_MD_CONFIG_T *pMdDefault, const TRDP_MEM_CONFIG_T *pMemConfig, TRDP_OPTION_T option)

  Initialize the TRDP stack.

- TRDP_ERR_T tlc_terminate (TRDP_APP_SESSION_T appHandle)

  Un-Initialize Clean up when app quits.

- TRDP_ERR_T tlc_reinit (TRDP_APP_SESSION_T appHandle)

  Re-Initialize Should be called by the application when a link-down/link-up event has occured during normal operation.

- const char * tlc_getVersion (void)

  Return a human readable version representation.

- TRDP_ERR_T tlp_setRedundant (TRDP_APP_SESSION_T appHandle, UINT32 redId, BOOL leader)

  Do not send non-redundant PDs when we are follower.

- EXT_DECL TRDP_ERR_T tlp_getRedundant (TRDP_APP_SESSION_T appHandle, UINT32 redId, BOOL *pLeader)

  Get status of redundant ComIds.

- void tlc_setTopoCount (UINT32 topoCount)

  Set new topocount for trainwide communication.

- EXT_DECL TRDP_ERR_T tlp_publish (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T *pPubHandle, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, UINT32 interval, UINT32 redId, TRDP_FLAGS_T pktFlags, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, BOOL subs, UINT16 offsetAddress)

  Prepare for sending PD messages.

- TRDP_ERR_T tlp_unpublish (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T pubHandle)

  Stop sending PD messages.

- TRDP_ERR_T tlp_put (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T pubHandle, const UINT8 *pData, UINT32 dataSize)

  Update the process data to send.

- EXT_DECL TRDP_ERR_T tlc_getInterval (TRDP_APP_SESSION_T appHandle, TRDP_TIME_T *pInterval, TRDP_FDS_T *pFileDesc, INT32 *pNoDesc)

  Get the lowest time interval for PDs.

- EXT_DECL TRDP_ERR_T tlc_process (TRDP_APP_SESSION_T appHandle, TRDP_FDS_T *pRfds, INT32 *pCount)

  Work loop of the TRDP handler.

- EXT_DECL TRDP_ERR_T tlp_subscribe (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T *pSubHandle, const void *pUserRef, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr1, TRDP_IP_ADDR_T srcIpAddr2, TRDP_IP_ADDR_T destIpAddr, UINT32 timeout, TRDP_TO_BEHAVIOR_T toBehavior, UINT32 maxDataSize)

  Prepare for receiving PD messages.

- EXT_DECL TRDP_ERR_T tlp_unsubscribe (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T subHandle)

  Stop receiving PD messages.

- EXT_DECL TRDP_ERR_T tlp_get (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T subHandle, TRDP_FLAGS_T pktFlags, TRDP_PD_INFO_T *pPdInfo, UINT8 *pData, UINT32 *pDataSize)

  Get the last valid PD message.

5.8.1 Detailed Description

Functions for ECN communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_if.c 3 2012-06-04 12:52:54Z 97025

5.8.2 Function Documentation

5.8.2.1 EXT_DECL TRDP_ERR_T tlc_getInterval (TRDP_APP_SESSION_T *appHandle*, TRDP_TIME_T * *pInterval*, TRDP_FDS_T * *pFileDesc*, INT32 * *pNoDesc*)

Get the lowest time interval for PDs.

Return the maximum time interval suitable for 'select()' so that we can send due PD packets in time. If the PD send queue is empty, return zero time

Parameters:

← ***appHandle*** The handle returned by tlc_init

→ ***pInterval*** pointer to needed interval

↔ ***pFileDesc*** pointer to file descriptor set

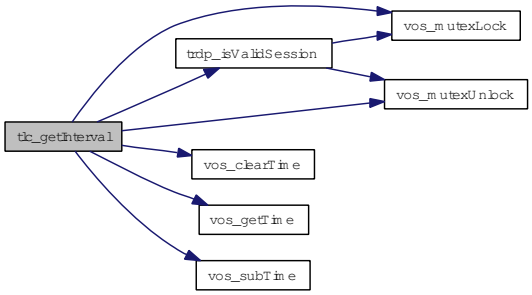
→ ***pNoDesc*** pointer to put no of used descriptors (for select())

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.2 const char* tlc_getVersion (void)

Return a human readable version representation.

Return string in the form 'v.r.u.b'

Return values:

const string

5.8.2.3 EXT_DECL TRDP_ERR_T tlc_init (TRDP_APP_SESSION_T * *pAppHandle*, TRDP_IP_ADDR_T *ownIpAddr*, TRDP_IP_ADDR_T *leaderIpAddr*, const TRDP_PRINT_DBG_T *pPrintDebugString*, const TRDP_MARSHALL_CONFIG_T * *pMarshall*, const TRDP_PD_CONFIG_T * *pPdDefault*, const TRDP_MD_CONFIG_T * *pMdDefault*, const TRDP_MEM_CONFIG_T * *pMemConfig*, TRDP_OPTION_T *option*)

Initialize the TRDP stack.

tlc_init returns in pAppHandle a unique handle to be used in further calls to the stack.

Parameters:

→ ***pAppHandle*** A handle for further calls to the trdp stack

← ***ownIpAddr*** Own IP address, can be different for each process in multiprocessing systems

← ***leaderIpAddr*** IP address of redundancy leader

← ***pPrintDebugString*** Pointer to debug print function

← ***pMarshall*** Pointer to marshalling configuration

← ***pPdDefault*** Pointer to default PD configuration

← ***pMdDefault*** Pointer to default MD configuration

← ***pMemConfig*** Pointer to memory configuration

← ***option*** options for library behavior

Return values:

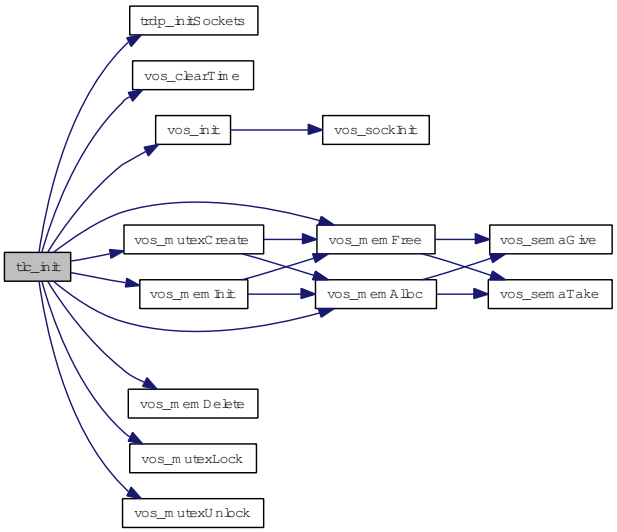
TRDP_NO_ERR no error

TRDP_MEM_ERR memory allocation failed

TRDP_PARAM_ERR initialization error

TRDP_SOCK_ERR socket error

Here is the call graph for this function:



5.8.2.4 EXT_DECL TRDP_ERR_T tlc_process (TRDP_APP_SESSION_T *appHandle*, TRDP_FDS_T * *pRfds*, INT32 * *pCount*)

Work loop of the TRDP handler.

Search the queue for pending PDs to be sent Search the receive queue for pending PDs (time out)

Parameters:

← ***appHandle*** The handle returned by tlc_init

← ***pRfds*** pointer to set of ready descriptors

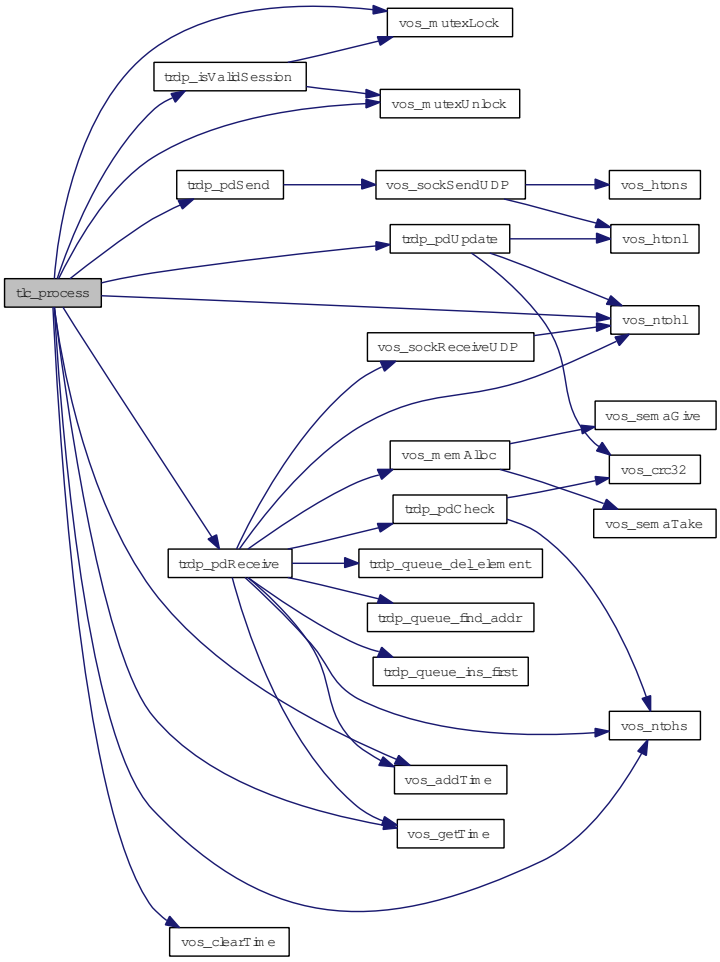
↔ ***pCount*** pointer to number of ready descriptors

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



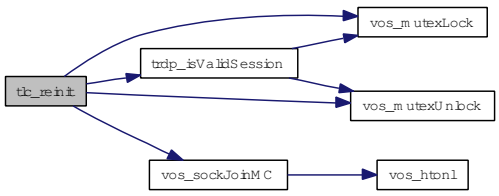
5.8.2.5 TRDP_ERR_T tlc_reinit (TRDP_APP_SESSION_T *appHandle*)

Re-Initialize Should be called by the application when a link-down/link-up event has occured during normal operation.

Re-Initialize.

We re-join

Here is the call graph for this function:



5.8.2.6 void tlc_setTopoCount (UINT32 *topoCount*)

Set new topocount for trainwide communication.

This value is used for validating outgoing and incoming packets only!

Parameters:

← ***topoCount*** New topoCount value

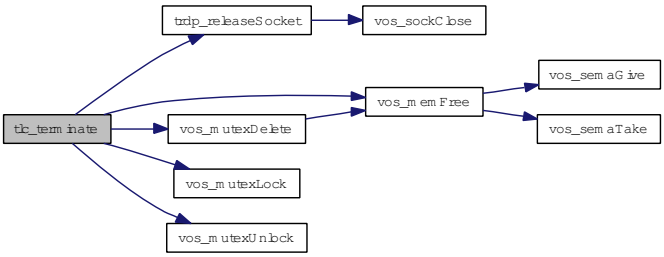
5.8.2.7 TRDP_ERR_T tlc_terminate (TRDP_APP_SESSION_T *appHandle*)

Un-Initialize Clean up when app quits.

Un-Initialize.

Mainly used for debugging/test runs

Here is the call graph for this function:



5.8.2.8 EXT_DECL TRDP_ERR_T tlp_get (TRDP_APP_SESSION_T *appHandle*, TRDP_SUB_T *subHandle*, TRDP_FLAGS_T *pktFlags*, TRDP_PD_INFO_T * *pPdInfo*, UINT8 * *pData*, UINT32 * *pDataSize*)

Get the last valid PD message.

This allows polling of PDs instead of event driven handling by callbacks

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***subHandle*** the handle returned by subscription

← ***pktFlags*** OPTION: TRDP_FLAGS_MARSHALL

↔ ***pPdInfo*** pointer to application's info buffer

↔ ***pData*** pointer to application's data buffer

↔ ***pDataSize*** in: size of buffer, out: size of data

Return values:

TRDP_NO_ERR no error

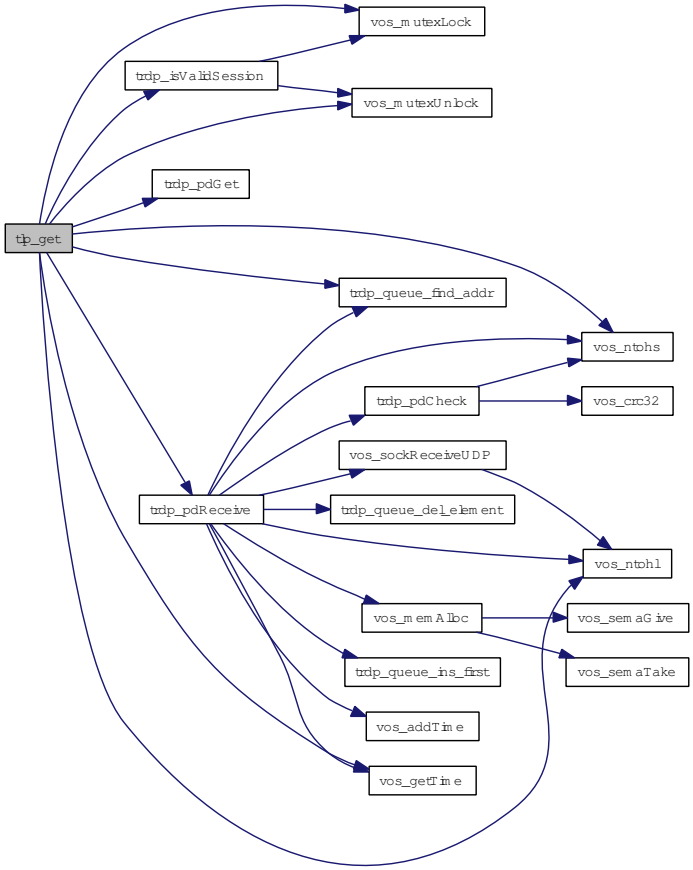
TRDP_PARAM_ERR parameter error

TRDP_SUB_ERR not subscribed

TRDP_TIMEOUT_ERR packet timed out

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.9 EXT_DECL TRDP_ERR_T tlp_getRedundant (TRDP_APP_SESSION_T *appHandle*, UINT32 *redId*, BOOL * *pLeader*)

Get status of redundant ComIds.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***redId*** will be returned for all ComID's with the given redId, 0 for all redId

↔ ***pLeader*** TRUE if we send (leader)

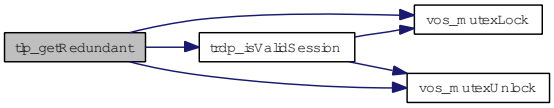
Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error / redId not existing

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.10 EXT_DECL TRDP_ERR_T tlp_publish (TRDP_APP_SESSION_T *appHandle*, TRDP_PUB_T * *pPubHandle*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *srcIpAddr*, TRDP_IP_ADDR_T *destIpAddr*, UINT32 *interval*, UINT32 *redId*, TRDP_FLAGS_T *pktFlags*, const TRDP_SEND_PARAM_T * *pSendParam*, const UINT8 * *pData*, UINT32 *dataSize*, BOOL *subs*, UINT16 *offsetAddress*)

Prepare for sending PD messages.

Queue a PD message, it will be send when trdp_work has been called

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pPubHandle*** returned handle for related unprepare

← ***comId*** comId of packet to send

← ***topoCount*** valid topocount, 0 for local consist

← ***srcIpAddr*** own IP address, 0 - srcIP will be set by the stack

← ***destIpAddr*** where to send the packet to

← ***interval*** frequency of PD packet (>= 10ms) in usec

← ***redId*** 0 - Non-redundant, > 0 valid redundancy group

← ***pktFlags*** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK

← ***pSendParam*** optional pointer to send parameter, NULL - default parameters are used

← ***pData*** pointer to packet data / dataset

← ***dataSize*** size of packet data <= 1436 without FCS

← ***subs*** substitution (Ladder)

← ***offsetAddress*** offset (Ladder)

Return values:

TRDP_NO_ERR no error

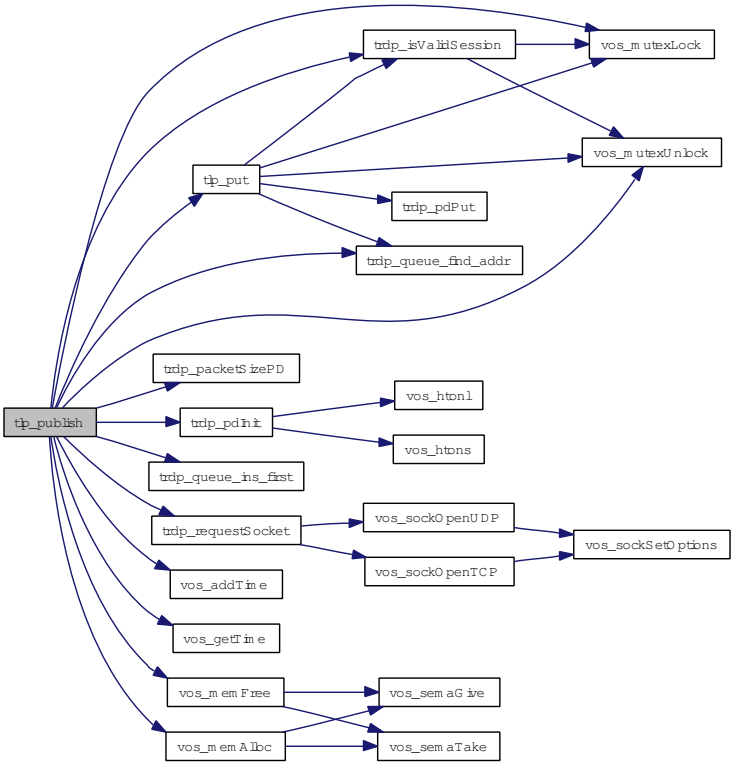
TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

TRDP_NOPUB_ERR Already published

Here is the call graph for this function:



5.8.2.11 TRDP_ERR_T tlp_put (TRDP_APP_SESSION_T *appHandle*, TRDP_PUB_T *pubHandle*, const UINT8 * *pData*, UINT32 *dataSize*)

Update the process data to send.

Update previously published data. The new telegram will be sent earliest when tlc_process is called.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***pubHandle*** the handle returned by publish

↔ ***pData*** pointer to application's data buffer

↔ ***dataSize*** size of data

Return values:

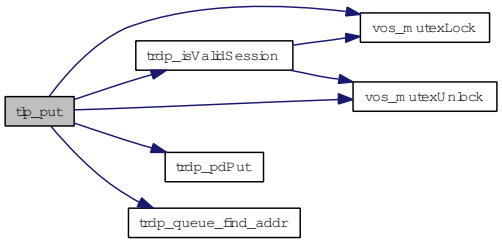
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOPUB_ERR not published

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.12 TRDP_ERR_T tlp_setRedundant (TRDP_APP_SESSION_T *appHandle*, UINT32 *redId*, BOOL *leader*)

Do not send non-redundant PDs when we are follower.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***redId*** will be set for all ComID's with the given redId, 0 to change for all redId

← ***leader*** TRUE if we send

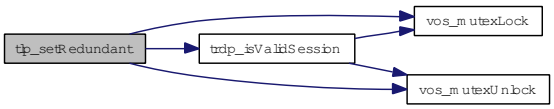
Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error / redId not existing

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.13 EXT_DECL TRDP_ERR_T tlp_subscribe (TRDP_APP_SESSION_T *appHandle*, TRDP_SUB_T * *pSubHandle*, const void * *pUserRef*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *srcIpAddr1*, TRDP_IP_ADDR_T *srcIpAddr2*, TRDP_IP_ADDR_T *destIpAddr*, UINT32 *timeout*, TRDP_TO_BEHAVIOR_T *toBehavior*, UINT32 *maxDataSize*)

Prepare for receiving PD messages.

Subscribe to a specific PD ComID and source IP To unsubscribe, set maxDataSize to zero!

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pSubHandle*** return a handle for these messages

← ***pUserRef*** user supplied value returned within the info structure

← ***comId*** comId of packet to receive

← ***topoCount*** valid topocount, 0 for local consist

← ***srcIpAddr1*** IP for source filtering, set 0 if not used

← ***srcIpAddr2*** Second source IP address for source filtering, set to zero if not used. Used e.g. for source filtering of redundant devices.

← ***destIpAddr*** IP address to join

← ***timeout*** timeout (>= 10ms) in usec

← ***toBehavior*** timeout behavior

← ***maxDataSize*** expected max. size of packet data

Return values:

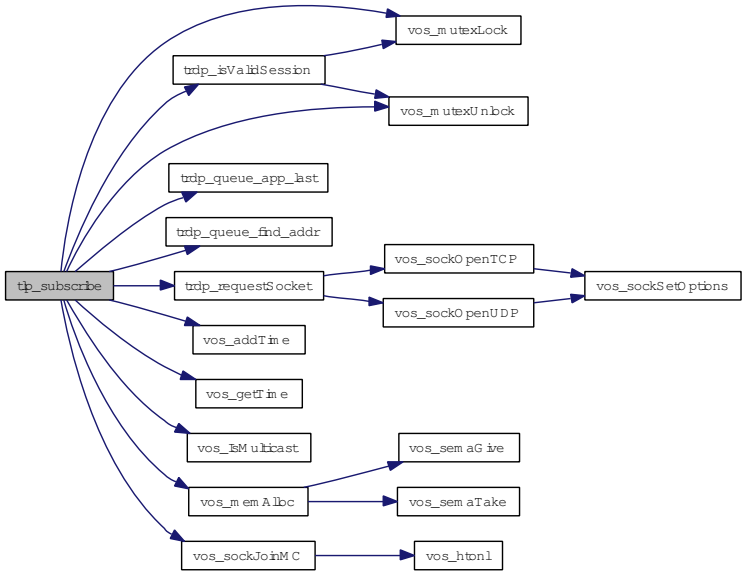
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not reserve memory (out of memory)

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.14 TRDP_ERR_T tlp_unpublish (TRDP_APP_SESSION_T *appHandle*, TRDP_PUB_T *pubHandle*)

Stop sending PD messages.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***pubHandle*** the handle returned by prepare

Return values:

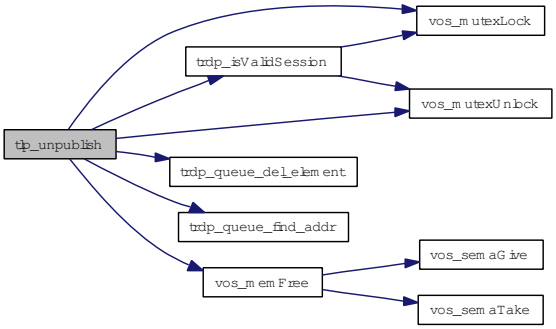
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOPUB_ERR not published

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.15 EXT_DECL TRDP_ERR_T tlp_unsubscribe (TRDP_APP_SESSION_T *appHandle*, TRDP_SUB_T *subHandle*)

Stop receiving PD messages.

Unsubscribe to a specific PD ComID

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***subHandle*** the handle returned by subscription

Return values:

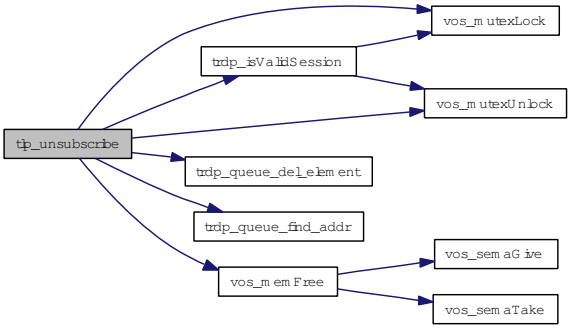
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_SUB_ERR not subscribed

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.16 BOOL trdp_isValidSession (TRDP_APP_SESSION_T *pSessionHandle*)

Check if the session handle is valid.

Parameters:

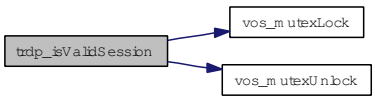
← ***pSessionHandle*** pointer to packet data (dataset)

Return values:

TRUE is valid

FALSE is invalid

Here is the call graph for this function:



5.8.2.17 TRDP_APP_SESSION_T* trdp_sessionQueue (void)

Get the session queue head pointer.

Return values:

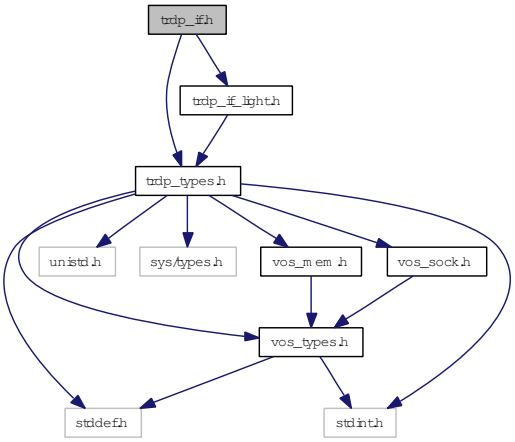
&sSession

5.9 trdp_if.h File Reference

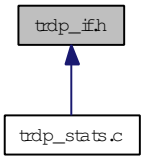
Typedefs for TRDP communication.

```
#include "trdp_types.h"
#include "trdp_if_light.h"
```

Include dependency graph for trdp_if.h:



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



Functions

- BOOL trdp_isValidSession (TRDP_APP_SESSION_T pSessionHandle)

  Check if the session handle is valid.

- TRDP_APP_SESSION_T * trdp_sessionQueue (void)

  Get the session queue head pointer.

5.9.1 Detailed Description

Typedefs for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_if.h 3 2012-06-04 12:52:54Z 97025

5.9.2 Function Documentation

5.9.2.1 BOOL trdp_isValidSession (TRDP_APP_SESSION_T *pSessionHandle*)

Check if the session handle is valid.

Parameters:

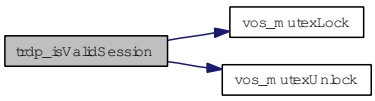
← ***pSessionHandle*** pointer to packet data (dataset)

Return values:

TRUE is valid

FALSE is invalid

Here is the call graph for this function:



5.9.2.2 TRDP_APP_SESSION_T* trdp_sessionQueue (void)

Get the session queue head pointer.

Return values:

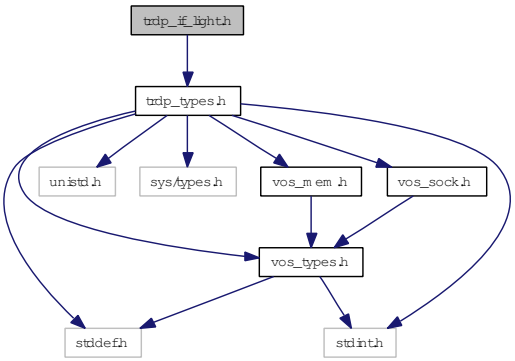
&sSession

5.10 trdp_if_light.h File Reference

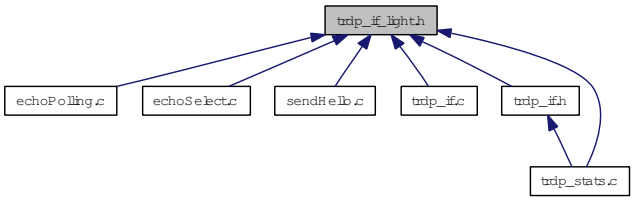
TRDP Light interface functions (API).

```
#include "trdp_types.h"
```

Include dependency graph for trdp_if_light.h:



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



Defines

- #define MD_SUPPORT 1

  Support for message data can only be excluded during compile time!

Functions

- EXT_DECL TRDP_ERR_T tlc_init (TRDP_APP_SESSION_T *pAppHandle, TRDP_IP_ADDR_T ownIpAddr, TRDP_IP_ADDR_T leaderIpAddr, const TRDP_PRINT_DBG_T pPrintDebugString, const TRDP_MARSHALL_CONFIG_T *pMarshall, const TRDP_PD_CONFIG_T *pPdDefault, const TRDP_MD_CONFIG_T *pMdDefault, const TRDP_MEM_CONFIG_T *pMemConfig, TRDP_OPTION_T option)

  Initialize the TRDP stack.

- EXT_DECL TRDP_ERR_T tlc_reinit (TRDP_APP_SESSION_T appHandle)

  Re-Initialize.

- EXT_DECL TRDP_ERR_T tlc_terminate (TRDP_APP_SESSION_T appHandle)

  Un-Initialize.

- EXT_DECL void tlc_setTopoCount (UINT32 topoCount)

  Set new topocount for trainwide communication.

- EXT_DECL TRDP_ERR_T tlc_freeBuf (TRDP_APP_SESSION_T appHandle, char *pBuf)

  Frees the buffer reserved by the TRDP layer.

- EXT_DECL TRDP_ERR_T tlc_getInterval (TRDP_APP_SESSION_T appHandle, TRDP_TIME_T *pInterval, TRDP_FDS_T *pFileDesc, INT32 *pNoDesc)

  Get the lowest time interval for PDs.

- EXT_DECL TRDP_ERR_T tlc_process (TRDP_APP_SESSION_T appHandle, TRDP_FDS_T *pRfds, INT32 *pCount)

  Work loop of the TRDP handler.

- EXT_DECL TRDP_ERR_T tlp_publish (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T *pPubHandle, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, UINT32 interval, UINT32 redId, TRDP_FLAGS_T pktFlags, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, BOOL subs, UINT16 offsetAddress)

  Prepare for sending PD messages.

- EXT_DECL TRDP_ERR_T tlp_unpublish (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T pubHandle)

  Stop sending PD messages.

- EXT_DECL TRDP_ERR_T tlp_put (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T pubHandle, const UINT8 *pData, UINT32 dataSize)

  Update the process data to send.

- EXT_DECL TRDP_ERR_T tlp_setRedundant (TRDP_APP_SESSION_T appHandle, UINT32 redId, BOOL leader)

  Do not send non-redundant PDs when we are follower.

- EXT_DECL TRDP_ERR_T tlp_getRedundant (TRDP_APP_SESSION_T appHandle, UINT32 redId, BOOL *pLeader)

  Get status of redundant ComIds.

- EXT_DECL TRDP_ERR_T tlp_request (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T subHandle, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, UINT32 redId, TRDP_FLAGS_T pktFlags, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, UINT32 replyComId, TRDP_IP_ADDR_T replyIpAddr, BOOL subs, UINT16 offsetAddr)

  Initiate sending PD messages (PULL).

- EXT_DECL TRDP_ERR_T tlp_subscribe (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T *pSubHandle, const void *pUserRef, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr1, TRDP_IP_ADDR_T srcIpAddr2, TRDP_IP_ADDR_T destIpAddr, UINT32 timeout, TRDP_TO_BEHAVIOR_T toBehavior, UINT32 maxDataSize)

  Prepare for receiving PD messages.

- EXT_DECL TRDP_ERR_T tlp_unsubscribe (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T subHandle)

  Stop receiving PD messages.

- EXT_DECL TRDP_ERR_T tlp_get (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T subHandle, TRDP_FLAGS_T pktFlags, TRDP_PD_INFO_T *pPdInfo, UINT8 *pData, UINT32 *pDataSize)

  Get the last valid PD message.

- EXT_DECL TRDP_ERR_T tlm_notify (TRDP_APP_SESSION_T appHandle, const void *pUserRef, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, const TRDP_URI_USER_T sourceURI, const TRDP_URI_USER_T destURI)

  Initiate sending MD notification message.

- EXT_DECL TRDP_ERR_T tlm_request (TRDP_APP_SESSION_T appHandle, const void *pUserRef, TRDP_UUID_T *pSessionId, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, UINT32 noOfRepliers, UINT32 replyTimeout, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, const TRDP_URI_USER_T srcURI, const TRDP_URI_USER_T destURI)

  Initiate sending MD request message.

- EXT_DECL TRDP_ERR_T tlm_confirm (TRDP_APP_SESSION_T appHandle, const void *pUserRef, const TRDP_UUID_T *pSessionId, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, UINT16 userStatus, TRDP_REPLY_STATUS_T replyStatus, const TRDP_SEND_PARAM_T *pSendParam, const TRDP_URI_USER_T srcURI, const TRDP_URI_USER_T destURI)

  Initiate sending MD confirm message.

- EXT_DECL TRDP_ERR_T tlm_abortSession (TRDP_APP_SESSION_T appHandle, TRDP_UUID_T *pSessionId)

  Cancel an open session.

- EXT_DECL TRDP_ERR_T tlm_addListener (TRDP_APP_SESSION_T appHandle, UINT32 *pListenHandle, const void *pUserRef, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, const TRDP_URI_USER_T destURI)

  Subscribe to MD messages.

- EXT_DECL TRDP_ERR_T tlm_delListener (TRDP_APP_SESSION_T appHandle, UINT32 listenHandle)

  Remove Listener.

- EXT_DECL TRDP_ERR_T tlm_reply (TRDP_APP_SESSION_T appHandle, TRDP_MSG_T msgType, void *pUserRef, TRDP_UUID_T *pSessionId, UINT32 topoCount, UINT32 comId, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, UINT16 userStatus, TRDP_REPLY_STATUS_T replyState, UINT32 replyTimeout, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, const TRDP_URI_USER_T srcURI, const TRDP_URI_USER_T destURI)

  Send a MD reply message.

- EXT_DECL const CHAR8 * tlc_getVersion (void)

Return a human readable version representation.

- EXT_DECL TRDP_ERR_T tlc_getStatistics (TRDP_APP_SESSION_T appHandle, TRDP_STATISTICS_T **ppStatistics)

  Return statistics.

- EXT_DECL TRDP_ERR_T tlc_getSubsStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumSubs, TRDP_SUBS_STATISTICS_T **ppStatistics)

  Return PD subscription statistics.

- EXT_DECL TRDP_ERR_T tlc_getPubStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumPub, TRDP_PUB_STATISTICS_T **ppStatistics)

  Return PD publish statistics.

- EXT_DECL TRDP_ERR_T tlc_getListStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumList, TRDP_LIST_STATISTICS_T **ppStatistics)

  Return MD listener statistics.

- EXT_DECL TRDP_ERR_T tlc_getRedStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumRed, TRDP_RED_STATISTICS_T **ppStatistics)

  Return redundancy group statistics.

- EXT_DECL TRDP_ERR_T tlc_getJoinStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumJoin, UINT32 **ppIpAddr)

  Return join statistics.

- EXT_DECL TRDP_ERR_T tlc_resetStatistics (TRDP_APP_SESSION_T appHandle)

  Reset statistics.

5.10.1 Detailed Description

TRDP Light interface functions (API).

Low level functions for communicating using the TRDP protocol

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_if_light.h 2 2012-06-04 11:25:16Z 97025

5.10.2 Function Documentation

5.10.2.1 EXT_DECL TRDP_ERR_T tlc_freeBuf (TRDP_APP_SESSION_T *appHandle*, char * *pBuf*)

Frees the buffer reserved by the TRDP layer.

Parameters:

← ***appHandle*** The handle returned by tlc_init

← ***pBuf*** pointer to the buffer to be freed

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR buffer pointer invalid

5.10.2.2 EXT_DECL TRDP_ERR_T tlc_getInterval (TRDP_APP_SESSION_T *appHandle*, TRDP_TIME_T * *pInterval*, TRDP_FDS_T * *pFileDesc*, INT32 * *pNoDesc*)

Get the lowest time interval for PDs.

Return the maximum time interval suitable for 'select()' so that we can send due PD packets in time. If the PD send queue is empty, return zero time

Parameters:

← ***appHandle*** The handle returned by tlc_init

→ ***pInterval*** pointer to needed interval

↔ ***pFileDesc*** pointer to file descriptor set

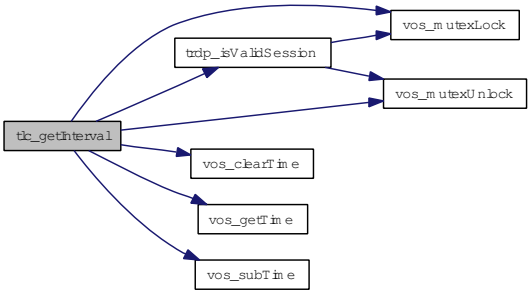
→ ***pNoDesc*** pointer to put no of used descriptors (for select())

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.3 EXT_DECL TRDP_ERR_T tlc_getJoinStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumJoin*, UINT32 ** *ppIpAddr*)

Return join statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumJoin*** Pointer to the number of joined IP Adresses

→ ***ppIpAddr*** Pointer to a list with the joined IP adresses

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:

5.10.2.4 EXT_DECL TRDP_ERR_T tlc_getListStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumList*, TRDP_LIST_STATISTICS_T ** *ppStatistics*)

Return MD listener statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumList*** Pointer to the number of listeners

→ ***ppStatistics*** Pointer to a list with the listener statistics information

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:

5.10.2.5 EXT_DECL TRDP_ERR_T tlc_getPubStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumPub*, TRDP_PUB_STATISTICS_T ** *ppStatistics*)

Return PD publish statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumPub*** Pointer to the number of publishers

→ ***ppStatistics*** Pointer to a list with the publish statistics information

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:

5.10.2.6 EXT_DECL TRDP_ERR_T tlc_getRedStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumRed*, TRDP_RED_STATISTICS_T ** *ppStatistics*)

Return redundancy group statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumRed*** Pointer to the number of redundancy groups

→ ***ppStatistics*** Pointer to a list with the redundancy group information

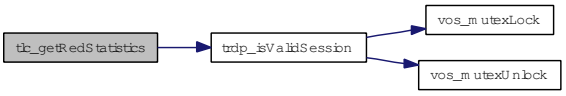
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.10.2.7 EXT_DECL TRDP_ERR_T tlc_getStatistics (TRDP_APP_SESSION_T *appHandle*, TRDP_STATISTICS_T ** *ppStatistics*)

Return statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***ppStatistics*** Statistics for this application session

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:

5.10.2.8 EXT_DECL TRDP_ERR_T tlc_getSubsStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumSubs*, TRDP_SUBS_STATISTICS_T ** *ppStatistics*)

Return PD subscription statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumSubs*** Pointer to the number of subscriptions

→ ***ppStatistics*** Pointer to a list with the subscription statistics information

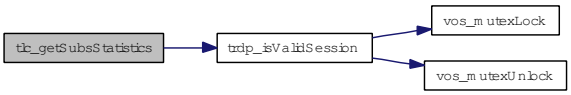
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.10.2.9 EXT_DECL const CHAR8* tlc_getVersion (void)

Return a human readable version representation.

Return string in the form 'v.r.u.b'

Return values:

const string

5.10.2.10 EXT_DECL TRDP_ERR_T tlc_init (TRDP_APP_SESSION_T * *pAppHandle*, TRDP_IP_ADDR_T *ownIpAddr*, TRDP_IP_ADDR_T *leaderIpAddr*, const TRDP_PRINT_DBG_T *pPrintDebugString*, const TRDP_MARSHALL_CONFIG_T * *pMarshall*, const TRDP_PD_CONFIG_T * *pPdDefault*, const TRDP_MD_CONFIG_T * *pMdDefault*, const TRDP_MEM_CONFIG_T * *pMemConfig*, TRDP_OPTION_T *option*)

Initialize the TRDP stack.

tlc_init returns in pAppHandle a unique handle to be used in further calls to the stack.

Parameters:

→ ***pAppHandle*** A handle for further calls to the trdp stack

← ***ownIpAddr*** Own IP address, can be different for each process in multiprocessing systems

← ***leaderIpAddr*** IP address of redundancy leader

← ***pPrintDebugString*** Pointer to debug print function

← ***pMarshall*** Pointer to marshalling configuration

← ***pPdDefault*** Pointer to default PD configuration

← ***pMdDefault*** Pointer to default MD configuration

← ***pMemConfig*** Pointer to memory configuration

← ***option*** options for library behavior

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR initialization error

TRDP_SOCK_ERR socket error

tlc_init returns in pAppHandle a unique handle to be used in further calls to the stack.

Parameters:

→ ***pAppHandle*** A handle for further calls to the trdp stack

← ***ownIpAddr*** Own IP address, can be different for each process in multiprocessing systems

← ***leaderIpAddr*** IP address of redundancy leader

← ***pPrintDebugString*** Pointer to debug print function

← ***pMarshall*** Pointer to marshalling configuration

← ***pPdDefault*** Pointer to default PD configuration

← ***pMdDefault*** Pointer to default MD configuration

← ***pMemConfig*** Pointer to memory configuration

← ***option*** options for library behavior

Return values:

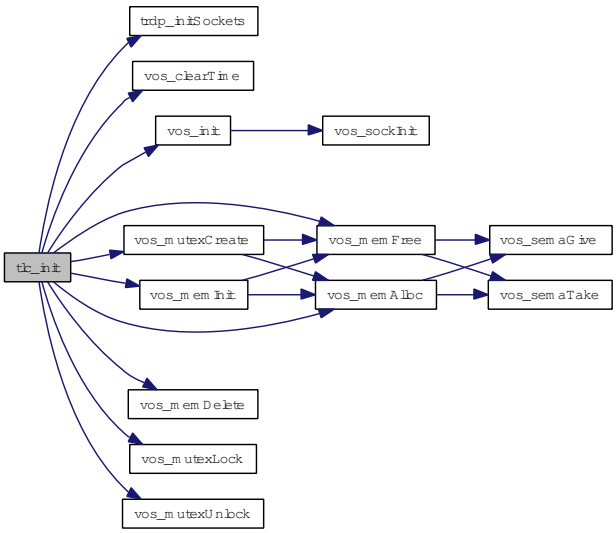
TRDP_NO_ERR no error

TRDP_MEM_ERR memory allocation failed

TRDP_PARAM_ERR initialization error

TRDP_SOCK_ERR socket error

Here is the call graph for this function:



5.10.2.11 EXT_DECL TRDP_ERR_T tlc_process (TRDP_APP_SESSION_T *appHandle*, TRDP_FDS_T * *pRfds*, INT32 * *pCount*)

Work loop of the TRDP handler.

Search the queue for pending PDs to be sent Search the receive queue for pending PDs (time out)

Parameters:

← ***appHandle*** The handle returned by tlc_init

← ***pRfds*** pointer to set of ready descriptors

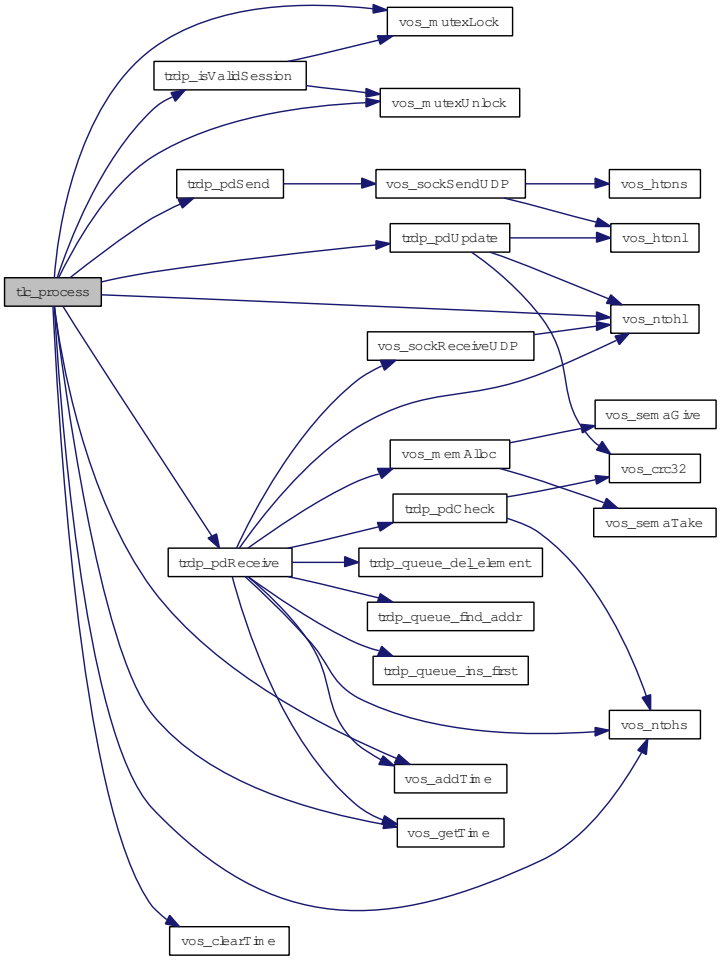
↔ ***pCount*** pointer to number of ready descriptors

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.12 EXT_DECL TRDP_ERR_T tlc_reinit (TRDP_APP_SESSION_T *appHandle*)

Re-Initialize.

Should be called by the application when a link-down/link-up event has occured during normal operation. We need to re-join the multicast groups...

Parameters:

← ***appHandle*** The handle returned by tlc_init

Return values:

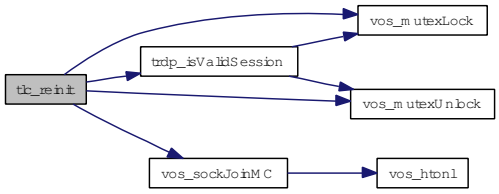
TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Re-Initialize.

We re-join

Here is the call graph for this function:



5.10.2.13 EXT_DECL TRDP_ERR_T tlc_resetStatistics (TRDP_APP_SESSION_T *appHandle*)

Reset statistics.

Parameters:

← ***appHandle*** the handle returned by tlc_init

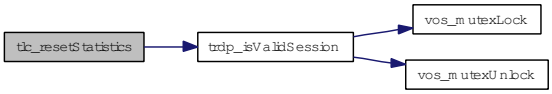
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.10.2.14 EXT_DECL void tlc_setTopoCount (UINT32 *topoCount*)

Set new topocount for trainwide communication.

This value is used for validating outgoing and incoming packets only!

Parameters:

← ***topoCount*** New topocount value

This value is used for validating outgoing and incoming packets only!

Parameters:

← ***topoCount*** New topoCount value

5.10.2.15 EXT_DECL TRDP_ERR_T tlc_terminate (TRDP_APP_SESSION_T *appHandle*)

Un-Initialize.

Clean up when app quits. Mainly used for debugging/test runs. No further calls to library allowed

Parameters:

← ***appHandle*** The handle returned by tlc_init

Return values:

TRDP_NO_ERR no error

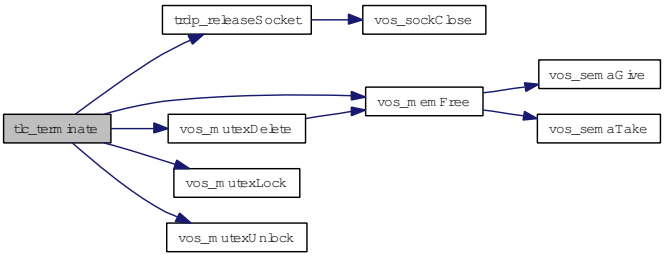
TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Un-Initialize.

Mainly used for debugging/test runs

Here is the call graph for this function:



5.10.2.16 EXT_DECL TRDP_ERR_T tlm_abortSession (TRDP_APP_SESSION_T *appHandle*, TRDP_UUID_T * *pSessionId*)

Cancel an open session.

Abort an open session; any pending messages will be dropped; session id set to zero

Parameters:

← ***appHandle*** the handle returned by tlc_init

↔ ***pSessionId*** Session ID returned by request

Return values:

TRDP_NO_ERR no error

TRDP_NO_SESSION_ERR no such session

TRDP_NOINIT_ERR handle invalid

5.10.2.17 EXT_DECL TRDP_ERR_T tlm_addListener (TRDP_APP_SESSION_T *appHandle*, UINT32 * *pListenHandle*, const void * *pUserRef*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *destIpAddr*, TRDP_FLAGS_T *pktFlags*, const TRDP_URI_USER_T *destURI*)

Subscribe to MD messages.

Add a listener to TRDP to get notified when messages are received

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pListenHandle*** Listener ID returned

← ***pUserRef*** user supplied value returned with reply

← ***comId*** comId to be observed

← ***topoCount*** topocount to use

← ***destIpAddr*** destination IP address

← ***pktFlags*** optional marshalling

← ***destURI*** only functional group of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NOINIT_ERR handle invalid

5.10.2.18 EXT_DECL TRDP_ERR_T tlm_confirm (TRDP_APP_SESSION_T *appHandle*, const void * *pUserRef*, const TRDP_UUID_T * *pSessionId*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *srcIpAddr*, TRDP_IP_ADDR_T *destIpAddr*, TRDP_FLAGS_T *pktFlags*, UINT16 *userStatus*, TRDP_REPLY_STATUS_T *replyStatus*, const TRDP_SEND_PARAM_T * *pSendParam*, const TRDP_URI_USER_T *srcURI*, const TRDP_URI_USER_T *destURI*)

Initiate sending MD confirm message.

Send a MD confirmation message

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***pUserRef*** user supplied value returned with reply

← ***pSessionId*** Session ID returned by request

← ***comId*** comId of packet to be sent

← ***topoCount*** topocount to use

← ***srcIpAddr*** own IP address, 0 - srcIP will be set by the stack

← ***destIpAddr*** where to send the packet to

← ***pktFlags*** OPTION: TRDP_FLAGS_CALLBACK

← ***userStatus*** Info for requester about application errors

← ***replyStatus*** Info for requester about stack errors

← ***pSendParam*** Pointer to send parameters, NULL to use default send parameters

← ***srcURI*** only functional group of source URI

← ***destURI*** only functional group of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NO_SESSION_ERR no such session

TRDP_NOINIT_ERR handle invalid

5.10.2.19 EXT_DECL TRDP_ERR_T tlm_delListener (TRDP_APP_SESSION_T *appHandle*, UINT32 *listenHandle*)

Remove Listener.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***listenHandle*** Listener ID returned

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOINIT_ERR handle invalid

5.10.2.20 EXT_DECL TRDP_ERR_T tlm_notify (TRDP_APP_SESSION_T *appHandle*, const void * *pUserRef*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *srcIpAddr*, TRDP_IP_ADDR_T *destIpAddr*, TRDP_FLAGS_T *pktFlags*, const TRDP_SEND_PARAM_T * *pSendParam*, const UINT8 * *pData*, UINT32 *dataSize*, const TRDP_URI_USER_T *sourceURI*, const TRDP_URI_USER_T *destURI*)

Initiate sending MD notification message.

Send a MD notification message

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***pUserRef*** user supplied value returned with reply

← ***comId*** comId of packet to be sent

← ***topoCount*** topocount to use

← ***srcIpAddr*** own IP address, 0 - srcIP will be set by the stack

← ***destIpAddr*** where to send the packet to

← ***pktFlags*** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK

← ***pSendParam*** optional pointer to send parameter, NULL - default parameters are used

← ***pData*** pointer to packet data / dataset

← ***dataSize*** size of packet data

← ***sourceURI*** only functional group of source URI

← ***destURI*** only functional group of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NOINIT_ERR handle invalid

5.10.2.21 EXT_DECL TRDP_ERR_T tlm_reply (TRDP_APP_SESSION_T *appHandle*, TRDP_MSG_T *msgType*, void * *pUserRef*, TRDP_UUID_T * *pSessionId*, UINT32 *topoCount*, UINT32 *comId*, TRDP_IP_ADDR_T *srcIpAddr*, TRDP_IP_ADDR_T *destIpAddr*, TRDP_FLAGS_T *pktFlags*, UINT16 *userStatus*, TRDP_REPLY_STATUS_T *replyState*, UINT32 *replyTimeout*, const TRDP_SEND_PARAM_T * *pSendParam*, const UINT8 * *pData*, UINT32 *dataSize*, const TRDP_URI_USER_T *srcURI*, const TRDP_URI_USER_T *destURI*)

Send a MD reply message.

Send a MD reply message after receiving an request

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***msgType*** Type of message: 'Mp', 'Me', or 'Mq'

← ***pUserRef*** user supplied value returned with reply

← ***pSessionId*** Session ID returned by indication

← ***topoCount*** topocount to use

← ***comId*** comId of packet to be sent

← ***srcIpAddr*** own IP address, 0 - srcIP will be set by the stack

← ***destIpAddr*** where to send the packet to

← ***pktFlags*** optional marshalling

← ***userStatus*** Info for requester about application errors

← ***replyState*** Info for requester about stack errors

← ***replyTimeout*** timeout for reply

← ***pSendParam*** Pointer to send parameters, NULL to use default send parameters

← ***pData*** pointer to packet data / dataset

← ***dataSize*** size of packet data

← ***srcURI*** only user part of source URI

← ***destURI*** only user part of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NO_SESSION_ERR no such session

TRDP_NOINIT_ERR handle invalid

5.10.2.22 EXT_DECL TRDP_ERR_T tlm_request (TRDP_APP_SESSION_T *appHandle*, const void * *pUserRef*, TRDP_UUID_T * *pSessionId*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *srcIpAddr*, TRDP_IP_ADDR_T *destIpAddr*, TRDP_FLAGS_T *pktFlags*, UINT32 *noOfRepliers*, UINT32 *replyTimeout*, const TRDP_SEND_PARAM_T * *pSendParam*, const UINT8 * *pData*, UINT32 *dataSize*, const TRDP_URI_USER_T *srcURI*, const TRDP_URI_USER_T *destURI*)

Initiate sending MD request message.

Send a MD request message

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***pUserRef*** user supplied value returned with reply

→ ***pSessionId*** return session ID

← ***comId*** comId of packet to be sent

← ***topoCount*** topocount to use

← ***srcIpAddr*** own IP address, 0 - srcIP will be set by the stack

← ***destIpAddr*** where to send the packet to

← ***pktFlags*** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK

← ***noOfRepliers*** number of expected repliers, 0 if unknown

← ***replyTimeout*** timeout for reply

← ***pSendParam*** Pointer to send parameters, NULL to use default send parameters

← ***pData*** pointer to packet data / dataset

← ***dataSize*** size of packet data

← ***srcURI*** only functional group of source URI

← ***destURI*** only functional group of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NOINIT_ERR handle invalid

5.10.2.23 EXT_DECL TRDP_ERR_T tlp_get (TRDP_APP_SESSION_T *appHandle*, TRDP_SUB_T *subHandle*, TRDP_FLAGS_T *pktFlags*, TRDP_PD_INFO_T * *pPdInfo*, UINT8 * *pData*, UINT32 * *pDataSize*)

Get the last valid PD message.

This allows polling of PDs instead of event driven handling by callback

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***subHandle*** the handle returned by subscription

← ***pktFlags*** OPTION: TRDP_FLAGS_MARSHALL

↔ ***pPdInfo*** pointer to application's info buffer

↔ ***pData*** pointer to application's data buffer

↔ ***pDataSize*** in: size of buffer, out: size of data

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_SUB_ERR not subscribed

TRDP_TIMEOUT_ERR packet timed out

TRDP_NOINIT_ERR handle invalid

This allows polling of PDs instead of event driven handling by callbacks

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***subHandle*** the handle returned by subscription

← ***pktFlags*** OPTION: TRDP_FLAGS_MARSHALL

↔ ***pPdInfo*** pointer to application's info buffer

↔ ***pData*** pointer to application's data buffer

↔ ***pDataSize*** in: size of buffer, out: size of data

Return values:

TRDP_NO_ERR no error

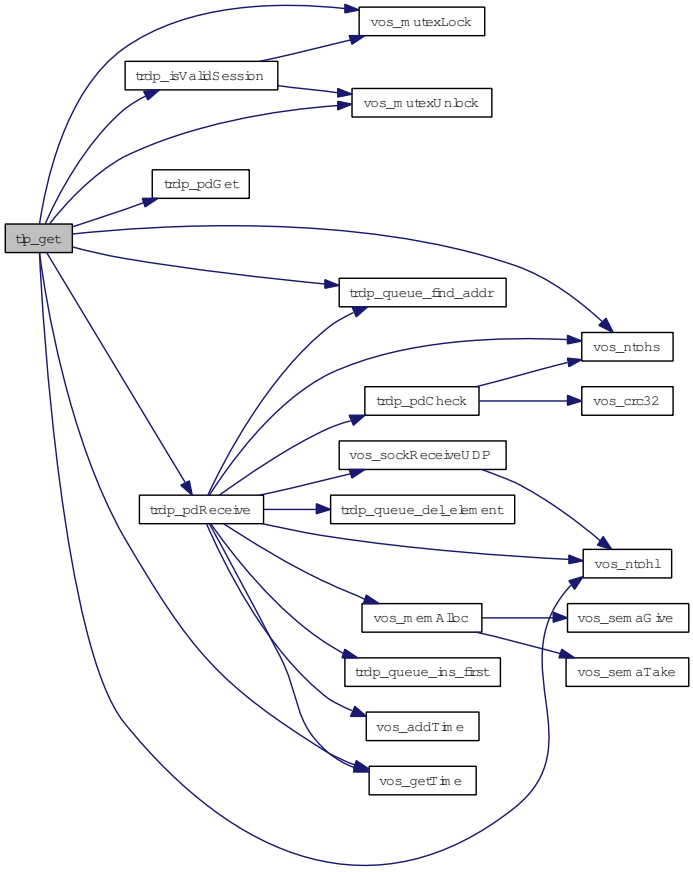
TRDP_PARAM_ERR parameter error

TRDP_SUB_ERR not subscribed

TRDP_TIMEOUT_ERR packet timed out

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.24 EXT_DECL TRDP_ERR_T tlp_getRedundant (TRDP_APP_SESSION_T *appHandle*, UINT32 *redId*, BOOL * *pLeader*)

Get status of redundant ComIds.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***redId*** will be set for all ComID's with the given redId, 0 for all redId

↔ ***pLeader*** TRUE if we send (leader)

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error / redId not existing

TRDP_NOINIT_ERR handle invalid

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***redId*** will be returned for all ComID's with the given redId, 0 for all redId

↔ ***pLeader*** TRUE if we send (leader)

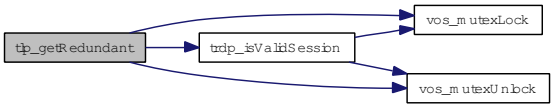
Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error / redId not existing

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.25 EXT_DECL TRDP_ERR_T tlp_publish (TRDP_APP_SESSION_T *appHandle*, TRDP_PUB_T * *pPubHandle*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *srcIpAddr*, TRDP_IP_ADDR_T *destIpAddr*, UINT32 *interval*, UINT32 *redId*, TRDP_FLAGS_T *pktFlags*, const TRDP_SEND_PARAM_T * *pSendParam*, const UINT8 * *pData*, UINT32 *dataSize*, BOOL *subs*, UINT16 *offsetAddress*)

Prepare for sending PD messages.

Queue a PD message, it will be send when trdp_work has been called

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pPubHandle*** returned handle for related unprepare

← ***comId*** comId of packet to send

← ***topoCount*** valid topocount, 0 for local consist

← ***srcIpAddr*** own IP address, 0 - srcIP will be set by the stack

← ***destIpAddr*** where to send the packet to

← ***interval*** frequency of PD packet (>= 10ms) in usec

← ***redId*** 0 - Non-redundant, > 0 valid redundancy group

← ***pktFlags*** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK

← ***pSendParam*** optional pointer to send parameter, NULL - default parameters are used

← ***pData*** pointer to packet data / dataset

← ***dataSize*** size of packet data

← ***subs*** substitution (Ladder)

← ***offsetAddress*** offset (Ladder)

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

Queue a PD message, it will be send when trdp_work has been called

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pPubHandle*** returned handle for related unprepare

← ***comId*** comId of packet to send

← ***topoCount*** valid topocount, 0 for local consist

← ***srcIpAddr*** own IP address, 0 - srcIP will be set by the stack

← ***destIpAddr*** where to send the packet to

← ***interval*** frequency of PD packet (>= 10ms) in usec

← ***redId*** 0 - Non-redundant, > 0 valid redundancy group

← ***pktFlags*** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK

← ***pSendParam*** optional pointer to send parameter, NULL - default parameters are used

← ***pData*** pointer to packet data / dataset

← ***dataSize*** size of packet data <= 1436 without FCS

← ***subs*** substitution (Ladder)

← ***offsetAddress*** offset (Ladder)

Return values:

TRDP_NO_ERR no error

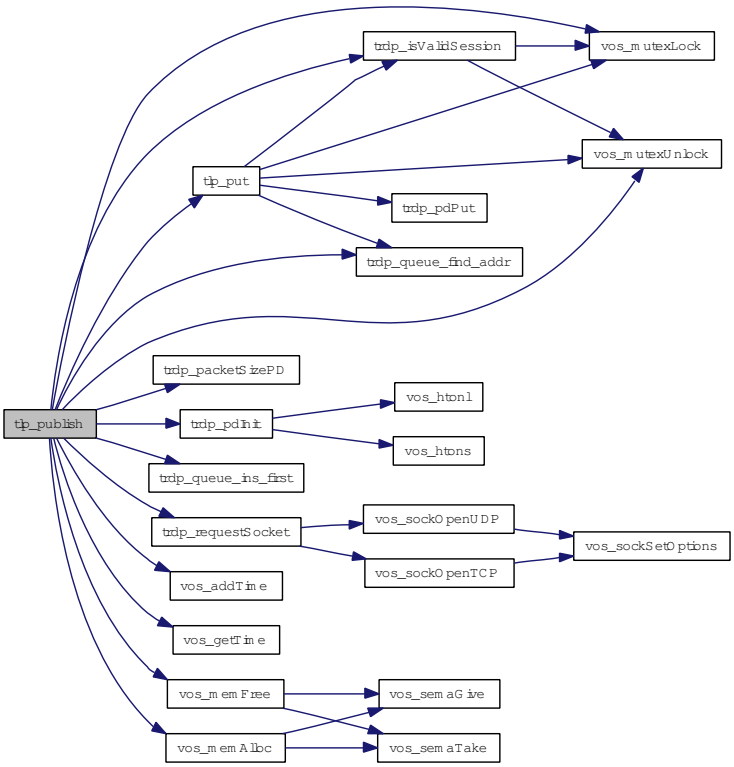
TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

TRDP_NOPUB_ERR Already published

Here is the call graph for this function:



5.10.2.26 EXT_DECL TRDP_ERR_T tlp_put (TRDP_APP_SESSION_T *appHandle*, TRDP_PUB_T *pubHandle*, const UINT8 * *pData*, UINT32 *dataSize*)

Update the process data to send.

Update previously published data. The new telegram will be sent earliest when tlc_process is called.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***pubHandle*** the handle returned by publish

↔ ***pData*** pointer to application's data buffer

↔ ***dataSize*** size of data

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_PUB_ERR not published

TRDP_NOINIT_ERR handle invalid

Update previously published data. The new telegram will be sent earliest when tlc_process is called.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***pubHandle*** the handle returned by publish

↔ ***pData*** pointer to application's data buffer

↔ ***dataSize*** size of data

Return values:

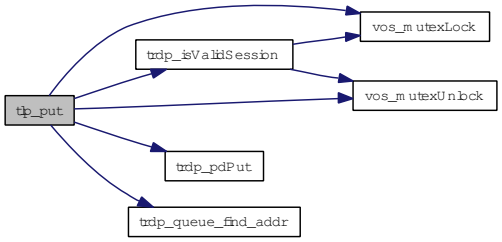
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOPUB_ERR not published

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.27 EXT_DECL TRDP_ERR_T tlp_request (TRDP_APP_SESSION_T *appHandle*, TRDP_SUB_T *subHandle*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *srcIpAddr*, TRDP_IP_ADDR_T *destIpAddr*, UINT32 *redId*, TRDP_FLAGS_T *pktFlags*, const TRDP_SEND_PARAM_T * *pSendParam*, const UINT8 * *pData*, UINT32 *dataSize*, UINT32 *replyComId*, TRDP_IP_ADDR_T *replyIpAddr*, BOOL *subs*, UINT16 *offsetAddr*)

Initiate sending PD messages (PULL).

Send a PD request message

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***subHandle*** handle from related subscribe

← ***comId*** comId of packet to be sent

← ***topoCount*** valid topocount, 0 for local consist

← ***srcIpAddr*** own IP address, 0 - srcIP will be set by the stack

← ***destIpAddr*** where to send the packet to

← ***redId*** 0 - Non-redundant, > 0 valid redundancy group

← ***pktFlags*** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK

← ***pSendParam*** optional pointer to send parameter, NULL - default parameters are used

← ***pData*** pointer to packet data / dataset

← ***dataSize*** size of packet data

← ***replyComId*** comId of reply

← ***replyIpAddr*** IP for reply

← ***subs*** substitution (Ladder)

← ***offsetAddr*** offset (Ladder)

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

5.10.2.28 EXT_DECL TRDP_ERR_T tlp_setRedundant (TRDP_APP_SESSION_T *appHandle*, UINT32 *redId*, BOOL *leader*)

Do not send non-redundant PDs when we are follower.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***redId*** will be set for all ComID's with the given redId, 0 to change for all redId

← ***leader*** TRUE if we send

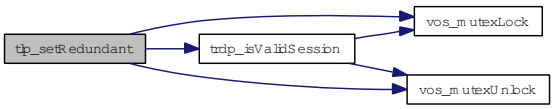
Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error / redId not existing

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.29 EXT_DECL TRDP_ERR_T tlp_subscribe (TRDP_APP_SESSION_T *appHandle*, TRDP_SUB_T * *pSubHandle*, const void * *pUserRef*, UINT32 *comId*, UINT32 *topoCount*, TRDP_IP_ADDR_T *srcIpAddr1*, TRDP_IP_ADDR_T *srcIpAddr2*, TRDP_IP_ADDR_T *destIpAddr*, UINT32 *timeout*, TRDP_TO_BEHAVIOR_T *toBehavior*, UINT32 *maxDataSize*)

Prepare for receiving PD messages.

Subscribe to a specific PD ComID and source IP To unsubscribe, set maxDataSize to zero!

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pSubHandle*** return a handle for these messages

← ***pUserRef*** user supplied value returned within the info structure

← ***comId*** comId of packet to receive

← ***topoCount*** valid topocount, 0 for local consist

← ***srcIpAddr1*** IP for source filtering, set 0 if not used

← ***srcIpAddr2*** Second source IP address for source filtering, set to zero if not used. Used e.g. for source filtering of redundant devices.

← ***destIpAddr*** IP address to join

← ***timeout*** timeout (>= 10ms) in usec

← ***toBehavior*** timeout behavior

← ***maxDataSize*** expected max. size of packet data

Return values:

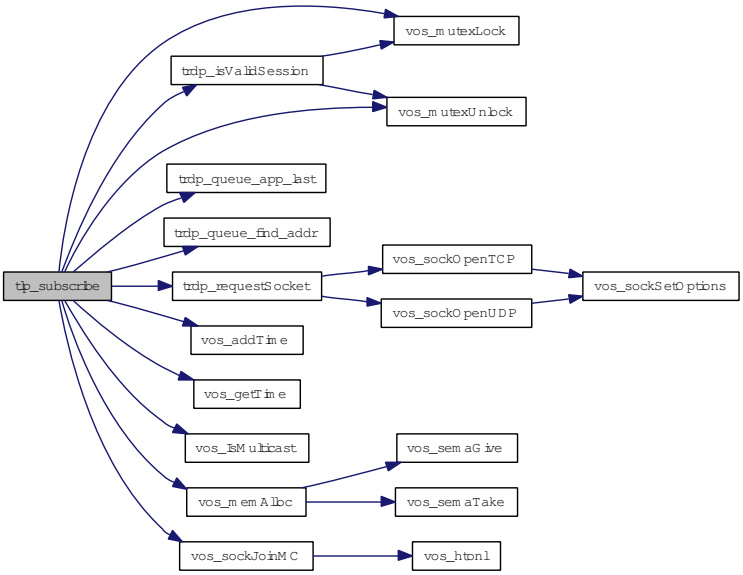
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not reserve memory (out of memory)

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.30 EXT_DECL TRDP_ERR_T tlp_unpublish (TRDP_APP_SESSION_T *appHandle*, TRDP_PUB_T *pubHandle*)

Stop sending PD messages.

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***pubHandle*** the handle returned by prepare

Return values:

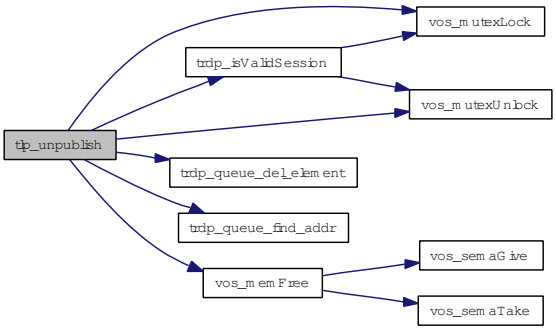
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOPUB_ERR not published

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.31 EXT_DECL TRDP_ERR_T tlp_unsubscribe (TRDP_APP_SESSION_T *appHandle*, TRDP_SUB_T *subHandle*)

Stop receiving PD messages.

Unsubscribe to a specific PD ComID

Parameters:

← ***appHandle*** the handle returned by tlc_init

← ***subHandle*** the handle returned by subscription

Return values:

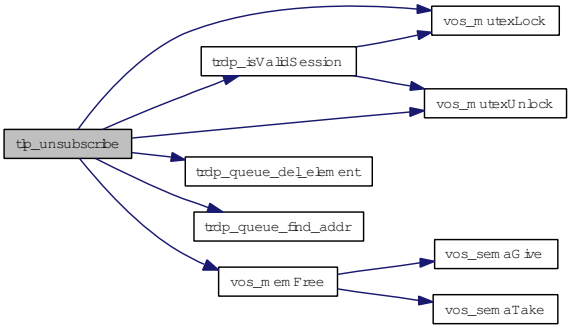
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_SUB_ERR not subscribed

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:

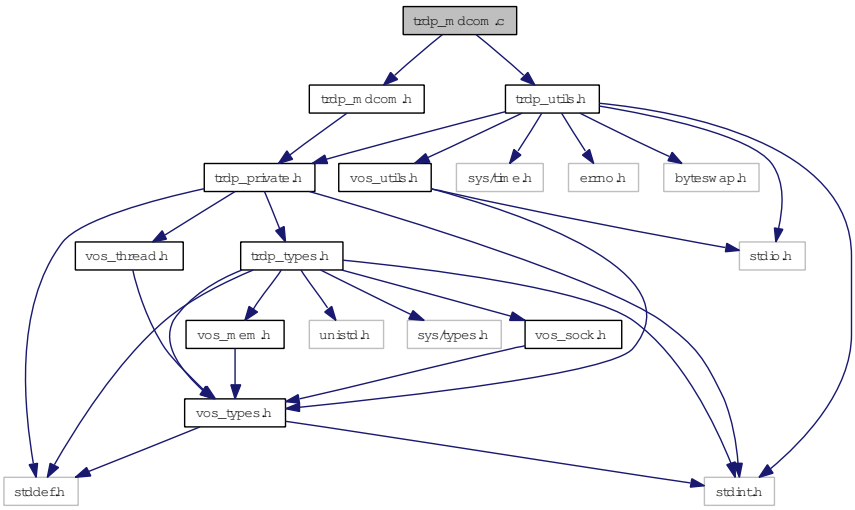


5.11 trdp_mdcom.c File Reference

Functions for MD communication.

```
#include "trdp_utils.h"
#include "trdp_mdcom.h"
```

Include dependency graph for trdp_mdcom.c:



Functions

- TRDP_ERR_T trdp_sendMD (int mdSock, const MD_ELE_T *pPacket)

  Send MD packet.

- TRDP_ERR_T trdp_rcvMD (int mdSock, MD_HEADER_T **ppPacket, ssize_t *pSize, uint32_t *pIPAddr)

  Receive MD packet.

5.11.1 Detailed Description

Functions for MD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_mdcom.c 2 2012-06-04 11:25:16Z 97025

5.11.2 Function Documentation

5.11.2.1 TRDP_ERR_T trdp_rcvMD (int *mdSock*, MD_HEADER_T ** *ppPacket*, ssize_t * *pSize*, uint32_t * *pIPAddr*)

Receive MD packet.

Parameters:

← ***mdSock*** socket descriptor

→ ***ppPacket*** pointer to pointer to received packet

→ ***pSize*** pointer to size of received packet

→ ***pIPAddr*** pointer to source IP address of packet

Return values:

TRDP_NO_ERR no error

TRDP_UNKNOWN_ERR error

5.11.2.2 TRDP_ERR_T trdp_sendMD (int *mdSock*, const MD_ELE_T * *pPacket*)

Send MD packet.

Parameters:

← ***mdSock*** socket descriptor

← ***pPacket*** pointer to packet to be sent

Return values:

TRDP_NO_ERR no error

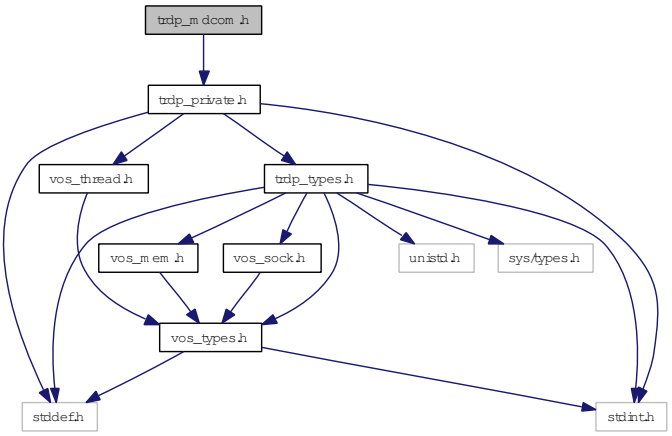
TRDP_UNKNOWN_ERR error

5.12 trdp_mdcom.h File Reference

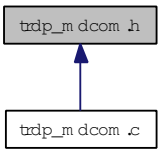
Functions for MD communication.

`#include "trdp_private.h"`

Include dependency graph for trdp_mdcom.h:



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



Functions

- TRDP_ERR_T trdp_sendMD (int sock, const MD_ELE_T *)
  Send MD packet.
- TRDP_ERR_T trdp_rcvMD (int sock, MD_HEADER_T **pPacket, ssize_t *pSize, uint32_t *pIPAddr)
  Receive MD packet.

5.12.1 Detailed Description

Functions for MD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_mdcom.h 2 2012-06-04 11:25:16Z 97025

5.12.2 Function Documentation

5.12.2.1 TRDP_ERR_T trdp_rcvMD (int *mdSock*, MD_HEADER_T ** *ppPacket*, ssize_t * *pSize*, uint32_t * *pIPAddr*)

Receive MD packet.

Parameters:

← ***mdSock*** socket descriptor

→ ***ppPacket*** pointer to pointer to received packet

→ ***pSize*** pointer to size of received packet

→ ***pIPAddr*** pointer to source IP address of packet

Return values:

TRDP_NO_ERR no error

TRDP_UNKNOWN_ERR error

5.12.2.2 TRDP_ERR_T trdp_sendMD (int *mdSock*, const MD_ELE_T * *pPacket*)

Send MD packet.

Parameters:

← ***mdSock*** socket descriptor

← ***pPacket*** pointer to packet to be sent

Return values:

TRDP_NO_ERR no error

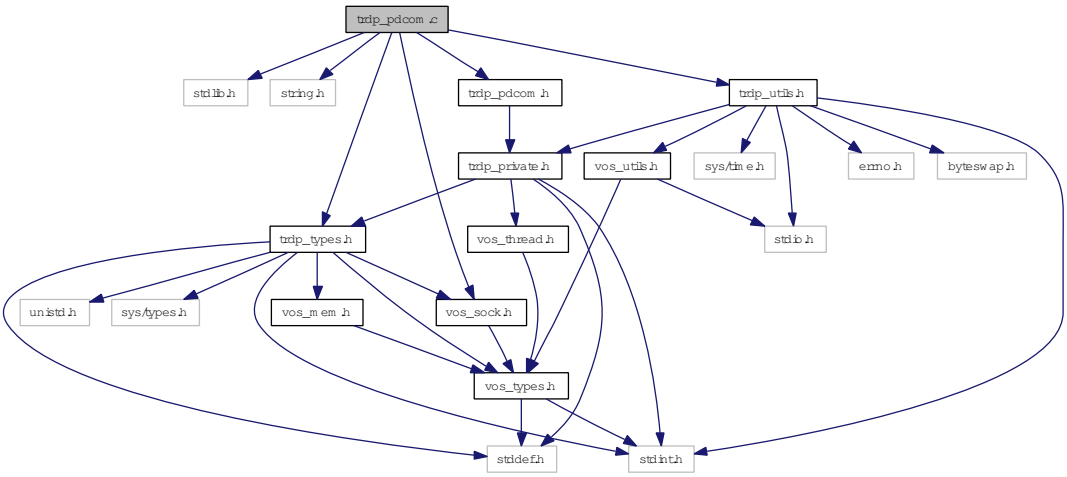
TRDP_UNKNOWN_ERR error

5.13 trdp_pdcom.c File Reference

Functions for PD communication.

```
#include <stdlib.h>
#include <string.h>
#include "trdp_types.h"
#include "trdp_utils.h"
#include "trdp_pdcom.h"
#include "vos_sock.h"
```

Include dependency graph for trdp_pdcom.c:



Functions

- void trdp_pdInit (PD_ELE_T *pPacket, TRDP_MSG_T type, UINT32 topoCount)

  Initialize/construct the packet Set the header infos.

- TRDP_ERR_T trdp_pdPut (PD_ELE_T *pPacket, TRDP_MARSHALL_T marshall, void *refCon, const UINT8 *pData, UINT32 dataSize)

  Copy data Set the header infos.

- TRDP_ERR_T trdp_pdGet (PD_ELE_T *pPacket, TRDP_UNMARSHALL_T unmarshall, void *refCon, const UINT8 *pData, UINT32 dataSize)

  Copy data Set the header infos.

- TRDP_ERR_T trdp_pdReceive (TRDP_SESSION_PT appHandle, INT32 sock)

  Receiving PD messages Read the receive socket for arriving PDs, copy the packet to a new PD_ELE_T Check for protocol errors and compare the received data to the data in our receive queue.

- void trdp_pdUpdate (PD_ELE_T *pPacket)

  Update the header values.

- TRDP_ERR_T trdp_pdCheck (PD_HEADER_T *pPacket, INT32 packetSize)

  Check if the PD header values are sane.

- TRDP_ERR_T trdp_pdSend (INT32 pdSock, const PD_ELE_T *pPacket)

  Send PD packet.

5.13.1 Detailed Description

Functions for PD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_pdcom.c 2 2012-06-04 11:25:16Z 97025

5.13.2 Function Documentation

5.13.2.1 TRDP_ERR_T trdp_pdCheck (PD_HEADER_T * *pPacket*, INT32 *packetSize*)

Check if the PD header values are sane.

Parameters:

← ***pPacket*** pointer to the packet to update

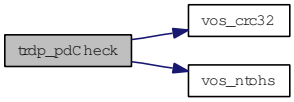
← ***packetSize*** max size to check

Return values:

TRDP_NO_ERR

TRDP_CRC_ERR

Here is the call graph for this function:



5.13.2.2 void trdp_pdInit (PD_ELE_T * *pPacket*, TRDP_MSG_T *type*, UINT32 *topoCount*)

Initialize/construct the packet Set the header infos.

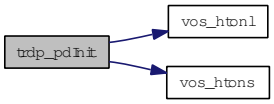
Parameters:

← ***pPacket*** pointer to the packet element to init

← ***type*** type the packet

← ***topoCount*** topocount to use for PD frame

Here is the call graph for this function:



5.13.2.3 TRDP_ERR_T trdp_pdReceive (TRDP_SESSION_PT *appHandle*, INT32 *sock*)

Receiving PD messages Read the receive socket for arriving PDs, copy the packet to a new PD_ELE_T Check for protocol errors and compare the received data to the data in our receive queue.

If it is a new packet, discard it (TBD: maybe for another session!). If it is an update, exchange the existing entry with the new one Call user's callback if needed

Parameters:

← ***appHandle*** session pointer

← ***sock*** the socket to read from

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

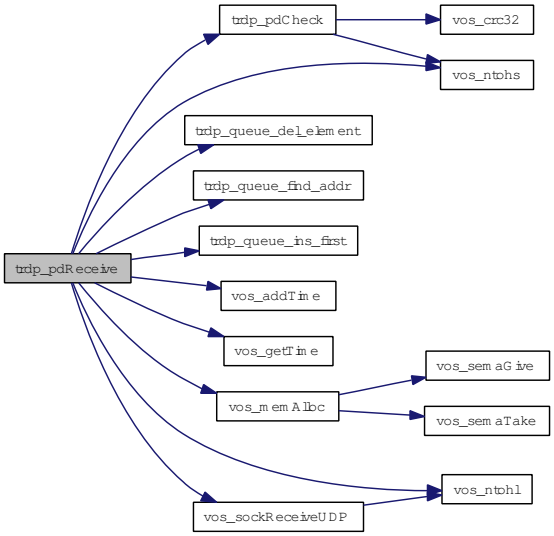
TRDP_WIRE_ERR protocol error (late packet, version mismatch)

TRDP_QUEUE_ERR not in queue

TRDP_CRC_ERR header checksum

TRDP_TOPOCOUNT_ERR invalid topocount

Here is the call graph for this function:



5.13.2.4 TRDP_ERR_T trdp_pdSend (INT32 *pdSock*, const PD_ELE_T * *pPacket*)

Send PD packet.

Parameters:

← ***pdSock*** socket descriptor

← ***pPacket*** pointer to packet to be sent

Return values:

!= NULL error

Here is the call graph for this function:

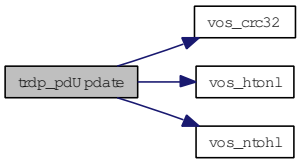
5.13.2.5 void trdp_pdUpdate (PD_ELE_T * *pPacket*)

Update the header values.

Parameters:

← ***pPacket*** pointer to the packet to update

Here is the call graph for this function:

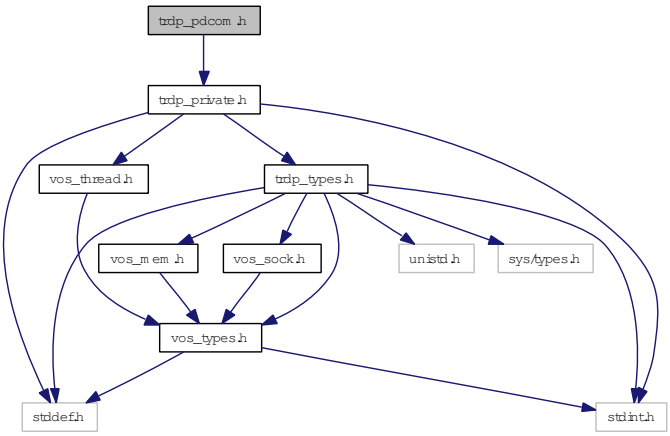


5.14 trdp_pdcom.h File Reference

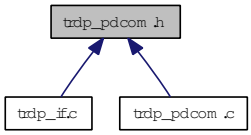
Functions for PD communication.

```
#include "trdp_private.h"
```

Include dependency graph for trdp_pdcom.h:



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



Functions

- void trdp_pdInit (PD_ELE_T *, TRDP_MSG_T, UINT32 topCount)

  Initialize/construct the packet Set the header infos.

- void trdp_pdUpdate (PD_ELE_T *)

  Update the header values.

- TRDP_ERR_T trdp_pdPut (PD_ELE_T *, TRDP_MARSHALL_T func, void *refCon, const UINT8 *pData, UINT32 dataSize)

  Copy data Set the header infos.

- TRDP_ERR_T trdp_pdCheck (PD_HEADER_T *pPacket, INT32 packetSize)

  Check if the PD header values are sane.

- TRDP_ERR_T trdp_pdSend (INT32 sock, const PD_ELE_T *)

  Send PD packet.

- TRDP_ERR_T trdp_pdGet (PD_ELE_T *pPacket, TRDP_UNMARSHALL_T unmarshall, void *refCon, const UINT8 *pData, UINT32 dataSize)

Copy data Set the header infos.

- TRDP_ERR_T trdp_pdReceive (TRDP_SESSION_PT pSessionHandle, INT32 sock)

  Receiving PD messages Read the receive socket for arriving PDs, copy the packet to a new PD_ELE_T Check for protocol errors and compare the received data to the data in our receive queue.

5.14.1 Detailed Description

Functions for PD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_pdcom.h 2 2012-06-04 11:25:16Z 97025

5.14.2 Function Documentation

5.14.2.1 TRDP_ERR_T trdp_pdCheck (PD_HEADER_T * *pPacket*, INT32 *packetSize*)

Check if the PD header values are sane.

Parameters:

← ***pPacket*** pointer to the packet to update

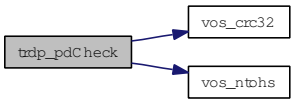
← ***packetSize*** max size to check

Return values:

TRDP_NO_ERR

TRDP_CRC_ERR

Here is the call graph for this function:



5.14.2.2 void trdp_pdInit (PD_ELE_T * *pPacket*, TRDP_MSG_T *type*, UINT32 *topoCount*)

Initialize/construct the packet Set the header infos.

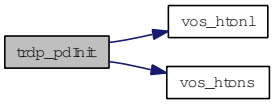
Parameters:

← ***pPacket*** pointer to the packet element to init

← ***type*** type the packet

← ***topoCount*** topocount to use for PD frame

Here is the call graph for this function:



5.14.2.3 TRDP_ERR_T trdp_pdReceive (TRDP_SESSION_PT *appHandle*, INT32 *sock*)

Receiving PD messages Read the receive socket for arriving PDs, copy the packet to a new PD_ELE_T Check for protocol errors and compare the received data to the data in our receive queue.

If it is a new packet, discard it (TBD: maybe for another session!). If it is an update, exchange the existing entry with the new one Call user's callback if needed

Parameters:

← ***appHandle*** session pointer

← ***sock*** the socket to read from

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

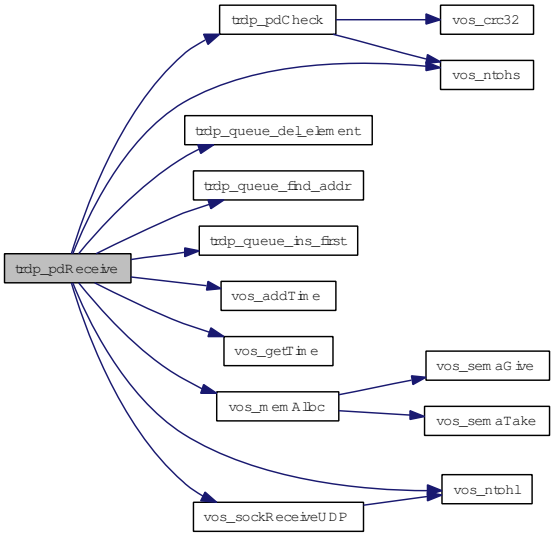
TRDP_WIRE_ERR protocol error (late packet, version mismatch)

TRDP_QUEUE_ERR not in queue

TRDP_CRC_ERR header checksum

TRDP_TOPOCOUNT_ERR invalid topocount

Here is the call graph for this function:



5.14.2.4 TRDP_ERR_T trdp_pdSend (INT32 *pdSock*, const PD_ELE_T * *pPacket*)

Send PD packet.

Parameters:

← ***pdSock*** socket descriptor

← ***pPacket*** pointer to packet to be sent

Return values:

!= NULL error

Here is the call graph for this function:

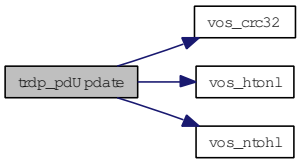
5.14.2.5 void trdp_pdUpdate (PD_ELE_T * *pPacket*)

Update the header values.

Parameters:

← ***pPacket*** pointer to the packet to update

Here is the call graph for this function:

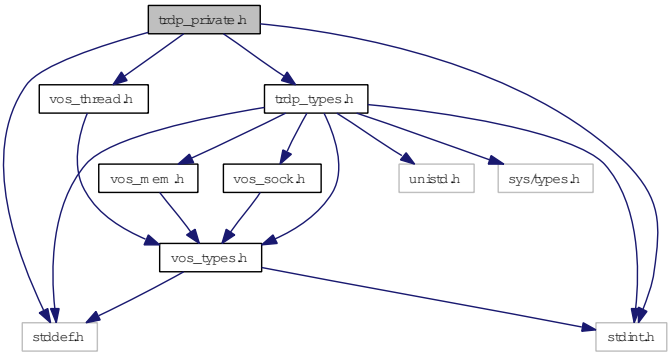


5.15 trdp_private.h File Reference

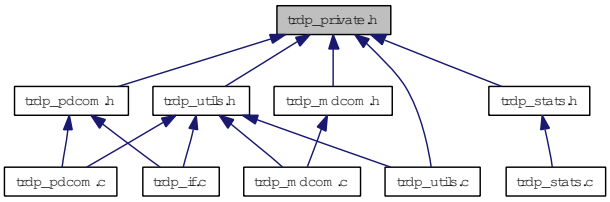
Typedefs for TRDP communication.

```
#include <stddef.h>
#include <stdint.h>
#include "trdp_types.h"
#include "vos_thread.h"
```

Include dependency graph for trdp_private.h:



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



Data Structures

- struct TRDP_HANDLE

  Hidden handle definition, used as unique addressing item.

- struct TRDP_SOCKETS

  Socket item.

- struct __attribute__

  TRDP process data header - network order and alignment.

- struct __attribute__

  TRDP process data header - network order and alignment.

- struct PD_ELE

  Queue element for PD packets to send or receive.

- struct MD_ELE

  Queue element for MD packets to send or receive or acknowledge.

- struct TRDP_SESSION

  Session/application variables store.

- struct TRDP_PD_STATISTICS

  Process data statistics.

- struct TRDP_MD_STATISTICS

  Message data statistics.

Defines

- #define IP_PD_UDP_PORT 20548

  process data UDP port

- #define IP_MD_UDP_PORT 20550

  message data UDP port

- #define IP_PD_PROTO_VER 0x0100

  Protocol version.

- #define ECHO_COMID 110

  comid used for echo

- #define TIMER_GRANULARITY 10000

  granularity in us

- #define MD_DEFAULT_REPLY_TIMEOUT 10000000

  default reply time out 10s

- #define MD_DEFAULT_CONFIRM_TIMEOUT 10000000

  default reply time out 10s

- #define MIN_PD_HEADER_SIZE sizeof(PD_HEADER_T)

  PD header size without FCS.

- #define ACK_TIME_OUT_VAL_DEF 500

  Default value in milliseconds for waiting on acknowledge message.

Typedefs

- typedef struct TRDP_HANDLE TRDP_ADDRESSES

  Hidden handle definition, used as unique addressing item.

- typedef struct TRDP_SOCKETS TRDP_SOCKETS_T
  Socket item.
- typedef struct PD_ELE PD_ELE_T
  Queue element for PD packets to send or receive.
- typedef struct MD_ELE MD_ELE_T
  Queue element for MD packets to send or receive or acknowledge.
- typedef struct TRDP_SESSION TRDP_SESSION_T
  Session/application variables store.
- typedef struct TRDP_PD_STATISTICS TRDP_PD_STATS_T
  Process data statistics.
- typedef struct TRDP_MD_STATISTICS TRDP_MD_STATS_T
  Message data statistics.

Enumerations

- enum TRDP_PRIV_FLAGS_T { , TRDP_TIMED_OUT = 0x2 }
  Internal flags for packets.
- enum TRDP_SOCK_TYPE_T {
  TRDP_SOCK_PD = 0,
  TRDP_SOCK_MD_UDP = 1,
  TRDP_SOCK_MD_TCP = 2 }
  Socket usage.

5.15.1 Detailed Description

Typedefs for TRDP communication.

TRDP internal type definitions

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_private.h 3 2012-06-04 12:52:54Z 97025

5.15.2 Enumeration Type Documentation

5.15.2.1 enum TRDP_PRIV_FLAGS_T

Internal flags for packets.

Enumerator:

TRDP_TIMED_OUT if set, informed the user

5.15.2.2 enum TRDP_SOCK_TYPE_T

Socket usage.

Enumerator:

TRDP_SOCK_PD Socket is used for UDP process data.

TRDP_SOCK_MD_UDP Socket is used for UDP message data.

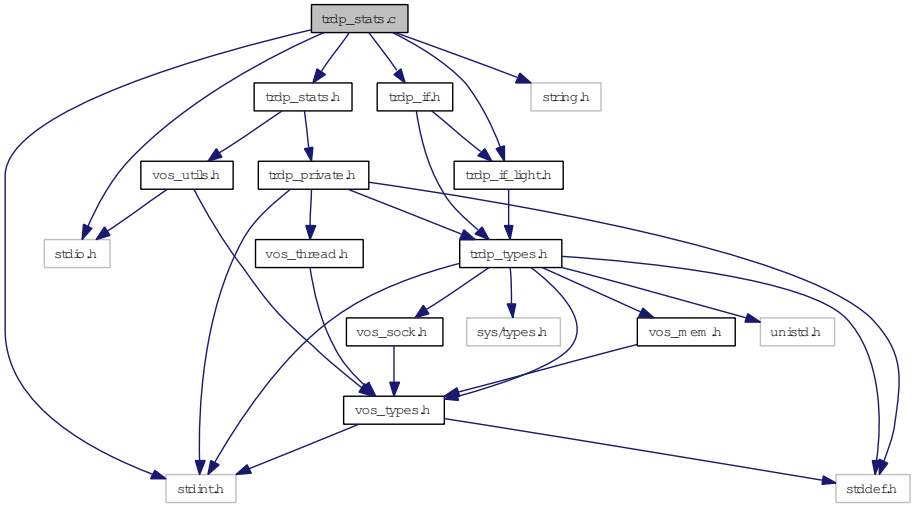
TRDP_SOCK_MD_TCP Socket is used for TCP message data.

5.16 trdp_stats.c File Reference

Statistics functions for TRDP communication.

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "trdp_stats.h"
#include "trdp_if_light.h"
#include "trdp_if.h"
```

Include dependency graph for trdp_stats.c:



Functions

- EXT_DECL TRDP_ERR_T tlc_getStatistics (TRDP_APP_SESSION_T appHandle, TRDP_STATISTICS_T **ppStatistics)

  Return statistics.

- EXT_DECL TRDP_ERR_T tlc_getSubsStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumSubs, TRDP_SUBS_STATISTICS_T **ppStatistics)

  Return PD subscription statistics.

- EXT_DECL TRDP_ERR_T tlc_getPubStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumPub, TRDP_PUB_STATISTICS_T **ppStatistics)

  Return PD publish statistics.

- EXT_DECL TRDP_ERR_T tlc_getListStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumList, TRDP_LIST_STATISTICS_T **ppStatistics)

  Return MD listener statistics.

- EXT_DECL TRDP_ERR_T tlc_getRedStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumRed, TRDP_RED_STATISTICS_T **ppStatistics)

Return redundancy group statistics.

- EXT_DECL TRDP_ERR_T tlc_getJoinStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumJoin, UINT32 **ppIpAddr)

  Return join statistics.

- EXT_DECL TRDP_ERR_T tlc_resetStatistics (TRDP_APP_SESSION_T appHandle)

  Reset statistics.

5.16.1 Detailed Description

Statistics functions for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_stats.c 3 2012-06-04 12:52:54Z 97025

5.16.2 Function Documentation

5.16.2.1 EXT_DECL TRDP_ERR_T tlc_getJoinStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumJoin*, UINT32 ** *ppIpAddr*)

Return join statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumJoin*** Pointer to the number of joined IP Adresses

→ ***ppIpAddr*** Pointer to a list with the joined IP adresses

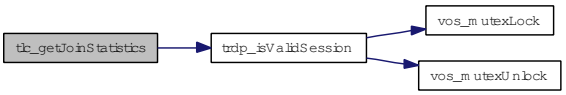
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.16.2.2 EXT_DECL TRDP_ERR_T tlc_getListStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumList*, TRDP_LIST_STATISTICS_T ** *ppStatistics*)

Return MD listener statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumList*** Pointer to the number of listeners

→ ***ppStatistics*** Pointer to a list with the listener statistics information

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:

5.16.2.3 EXT_DECL TRDP_ERR_T tlc_getPubStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumPub*, TRDP_PUB_STATISTICS_T ** *ppStatistics*)

Return PD publish statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumPub*** Pointer to the number of publishers

→ ***ppStatistics*** Pointer to a list with the publish statistics information

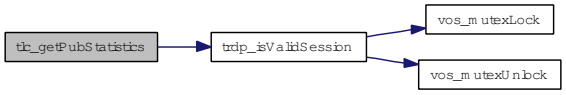
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.16.2.4 EXT_DECL TRDP_ERR_T tlc_getRedStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumRed*, TRDP_RED_STATISTICS_T ** *ppStatistics*)

Return redundancy group statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumRed*** Pointer to the number of redundancy groups

→ ***ppStatistics*** Pointer to a list with the redundancy group information

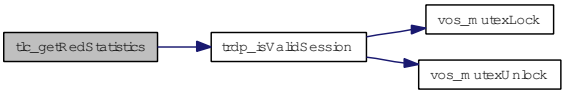
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.16.2.5 EXT_DECL TRDP_ERR_T tlc_getStatistics (TRDP_APP_SESSION_T *appHandle*, TRDP_STATISTICS_T ** *ppStatistics*)

Return statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***ppStatistics*** Statistics for this application session

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:

5.16.2.6 EXT_DECL TRDP_ERR_T tlc_getSubsStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumSubs*, TRDP_SUBS_STATISTICS_T ** *ppStatistics*)

Return PD subscription statistics.

Memory for statistics information will be reserved by tlc layer and needs to be freed by the user.

Parameters:

← ***appHandle*** the handle returned by tlc_init

→ ***pNumSubs*** Pointer to the number of subscriptions

→ ***ppStatistics*** Pointer to a list with the subscription statistics information

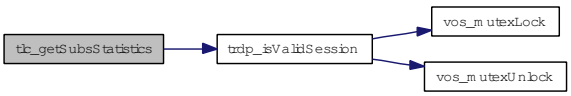
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.16.2.7 EXT_DECL TRDP_ERR_T tlc_resetStatistics (TRDP_APP_SESSION_T *appHandle*)

Reset statistics.

Parameters:

← ***appHandle*** the handle returned by tlc_init

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

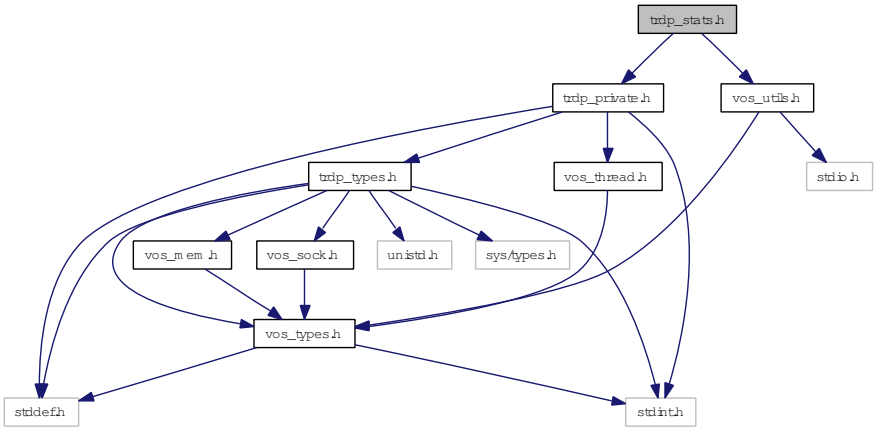
Here is the call graph for this function:

5.17 trdp_stats.h File Reference

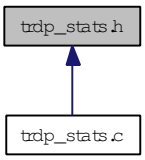
Statistics for TRDP communication.

```
#include "trdp_private.h"
#include "vos_utils.h"
```

Include dependency graph for trdp_stats.h:



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



5.17.1 Detailed Description

Statistics for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

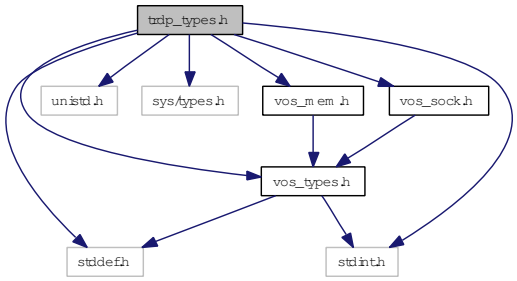
trdp_stats.h 3 2012-06-04 12:52:54Z 97025

5.18 trdp_types.h File Reference

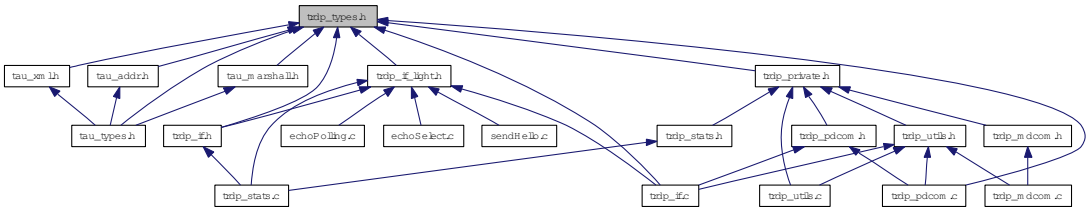
Typedefs for TRDP communication.

```
#include <stddef.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/types.h>
#include "vos_types.h"
#include "vos_mem.h"
#include "vos_sock.h"
```

Include dependency graph for trdp_types.h:



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



Data Structures

- struct TRDP_PD_INFO_T

  Process data info from received telegram; allows the application to generate responses.

- struct TRDP_MD_INFO_T

  Message data info from received telegram; allows the application to generate responses.

- struct TRDP_SEND_PARAM_T

  Quality/type of service and time to live.

- struct TRDP_DATASET_ELEMENT_T

  Dataset element definition.

- struct TRDP_DATASET_T

  Dataset definition.

- struct TRDP_MEM_STATISTICS_T

  TRDP statistics type definitions.

- struct TRDP_PD_STATISTICS_T

  Structure containing all general PD statistics information.

- struct TRDP_MD_STATISTICS_T

  Structure containing all general MD statistics information.

- struct TRDP_STATISTICS_T

  Structure containing all general memory, PD and MD statistics information.

- struct TRDP_SUBS_STATISTICS_T

  Table containing particular PD subscription information.

- struct TRDP_PUB_STATISTICS_T

  Table containing particular PD publishing information.

- struct TRDP_LIST_STATISTICS_T

  Information about a particular MD listener.

- struct TRDP_RED_STATISTICS_T

  A table containing PD redundant group information.

- struct TRDP_MARSHALL_CONFIG_T

  Marshaling/unmarshalling configuration.

- struct TRDP_PD_CONFIG_T

  Default PD configuration.

- struct TRDP_MD_CONFIG_T

  Default MD configuration.

- struct TRDP_MEM_CONFIG_T

  Structure describing memory (and its pre-fragmentation).

Defines

- #define TRDP_MAX_LABEL_LEN 16

  Maximum values.

- #define TRDP_MAX_URI_USER_LEN (2 ∗ TRDP_MAX_LABEL_LEN)

  URI user part incl.

- #define TRDP_MAX_URI_HOST_LEN (4 ∗ TRDP_MAX_LABEL_LEN)

URI host part length incl.

- #define TRDP_MAX_URI_LEN ((6 * TRDP_MAX_LABEL_LEN) + 8)
  URI length incl.
- #define TRDP_MAX_FILE_NAME_LEN 128
  path and file name length incl.
- #define USE_HEAP 0
  If this is set, we can allocate dynamically memory.

Typedefs

- typedef UINT32 TRDP_IP_ADDR_T
  TRDP general type definitions.
- typedef VOS_TIME_T TRDP_TIME_T
  Timer value compatible with timeval / select.
- typedef struct fd_set TRDP_FDS_T
  File descriptor set compatible with fd_set / select.
- typedef VOS_UUID_T TRDP_UUID_T
  UUID definition reuses the VOS definition.
- typedef VOS_PRINT_DBG_T TRDP_PRINT_DBG_T
  TRDP configuration type definitions.
- typedef VOS_LOG_T TRDP_LOG_T
  Categories for logging, reuse of the VOS definition.
- typedef TRDP_ERR_T(* TRDP_MARSHALL_T)(void *pRefCon, UINT32 comId, const UINT8 *pSrc, UINT8 *pDst, UINT32 *pDstSize)
  Function type for marshalling .
- typedef TRDP_ERR_T(* TRDP_UNMARSHALL_T)(void *pRefCon, UINT32 comId, const UINT8 *pSrc, UINT8 *pDst, UINT32 *pDstSize)
  Function type for unmarshalling.
- typedef void(* TRDP_PD_CALLBACK_T)(void *pRefCon, const TRDP_PD_INFO_T *pMsg, UINT8 *pData, UINT32 dataSize)
  Callback for receiving indications, timeouts, releases, responses.
- typedef void(* TRDP_MD_CALLBACK_T)(void *pRefCon, const TRDP_MD_INFO_T *pMsg, UINT8 *pData, UINT32 dataSize)
  Callback for receiving indications, timeouts, releases, responses.
- typedef VOS_MEM_BLK_T TRDP_MEM_BLK_T
  Enumeration type for memory pre-fragmentation, reuse of VOS definition.

Enumerations

- enum TRDP_ERR_T {
  TRDP_NO_ERR = 0,
  TRDP_PARAM_ERR = -1,
  TRDP_INIT_ERR = -2,
  TRDP_NOINIT_ERR = -3,
  TRDP_TIMEOUT_ERR = -4,
  TRDP_NODATA_ERR = -5,
  TRDP_SOCK_ERR = -6,
  TRDP_IO_ERR = -7,
  TRDP_MEM_ERR = -8,
  TRDP_SEMA_ERR = -9,
  TRDP_QUEUE_ERR = -10,
  TRDP_QUEUE_FULL_ERR = -11,
  TRDP_MUTEX_ERR = -12,
  TRDP_NOSESSION_ERR = -13,
  TRDP_SESSION_ABORT_ERR = -14,
  TRDP_NOSUB_ERR = -15,
  TRDP_NOPUB_ERR = -16,
  TRDP_NOLIST_ERR = -17,
  TRDP_CRC_ERR = -18 ,
  TRDP_TOPO_ERR = -20,
  TRDP_COMID_ERR = -21,
  TRDP_STATE_ERR = -22,
  TRDP_UNKNOWN_ERR = -99 }
  Return codes for all API functions.
- enum TRDP_MSG_T {
  TRDP_MSG_PD = 0x5064,
  TRDP_MSG_PR = 0x5072,
  TRDP_MSG_PE = 0x5065,
  TRDP_MSG_MN = 0x4D6E,
  TRDP_MSG_MR = 0x4D72,
  TRDP_MSG_MP = 0x4D70,
  TRDP_MSG_MQ = 0x4D71,
  TRDP_MSG_MC = 0x4D63,
  TRDP_MSG_ME = 0x4D65 }
  TRDP data transfer type definitions.
- enum TRDP_REPLY_STATUS_T
  Reply status messages.

- enum TRDP_FLAGS_T { ,
  TRDP_FLAGS_REDUNDANT = 0x1,
  TRDP_FLAGS_MARSHALL = 0x2,
  TRDP_FLAGS_CALLBACK = 0x4,
  TRDP_FLAGS_TCP = 0x8 }
  Various flags for PD and MD packets.
- enum TRDP_RED_STATE_T {
  TRDP_RED_FOLLOWER = 0,
  TRDP_RED_LEADER = 1 }
  Redundancy states.
- enum TRDP_TO_BEHAVIOR_T
  How invalid PD shall be handled.
- enum TRDP_DATA_TYPE_T {
  TRDP_BOOLEAN = -1,
  TRDP_CHAR8 = -2,
  TRDP_UTF16 = -3,
  TRDP_INT8 = -4,
  TRDP_INT16 = -5,
  TRDP_INT32 = -6,
  TRDP_INT64 = -7,
  TRDP_UINT8 = -8,
  TRDP_UINT16 = -9,
  TRDP_UINT32 = -10,
  TRDP_UINT64 = -11,
  TRDP_REAL32 = -12,
  TRDP_REAL64 = -13,
  TRDP_STRING = -14,
  TRDP_ARRAY = -15,
  TRDP_RECORD = -16,
  TRDP_TIMEDATE32 = -17,
  TRDP_TIMEDATE48 = -18,
  TRDP_TIMEDATE64 = -19 }
  TRDP dataset description definitions.
- enum TRDP_OPTION_T { ,
  TRDP_OPTION_BLOCK = 0x01,
  TRDP_OPTION_TRAFFIC_SHAPING = 0x02 }
  Various flags/general TRDP options for library initialization.

5.18.1 Detailed Description

Typedefs for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_types.h 2 2012-06-04 11:25:16Z 97025

5.18.2 Define Documentation

5.18.2.1 #define TRDP_MAX_FILE_NAME_LEN 128

path and file name length incl.

terminating '0'

5.18.2.2 #define TRDP_MAX_LABEL_LEN 16

Maximum values.

A uri is a string of the following form: trdp://[user part]@[host part] trdp://instLabel.funcLabel@devLabel.carLabel.cstLabel.trainLabel Hence the exact max. uri length is: 7 + (6 ∗ 15) + 5 ∗ (sizeof (separator)) + 1(terminating 0) to facilitate alignment the size will be increased by 1 byte label length incl. terminating '0'

5.18.2.3 #define TRDP_MAX_URI_HOST_LEN (4 ∗ TRDP_MAX_LABEL_LEN)

URI host part length incl.

terminating '0'

5.18.2.4 #define TRDP_MAX_URI_LEN ((6 ∗ TRDP_MAX_LABEL_LEN) + 8)

URI length incl.

terminating '0' and 1 padding byte

5.18.2.5 #define TRDP_MAX_URI_USER_LEN (2 ∗ TRDP_MAX_LABEL_LEN)

URI user part incl.

terminating '0'

5.18.3 Typedef Documentation

5.18.3.1 typedef UINT32 TRDP_IP_ADDR_T

TRDP general type definitions.

5.18.3.2 typedef TRDP_ERR_T(∗ TRDP_MARSHALL_T)(void ∗pRefCon, UINT32 comId, const UINT8 ∗pSrc, UINT8 ∗pDst, UINT32 ∗pDstSize)

Function type for marshalling .

The function must know about the dataset's alignment etc.

Parameters:

← ∗***pRefCon*** pointer to user context
← ***comId*** ComId to identify the structure out of a configuration
← ∗***pSrc*** pointer to received original message
← ∗***pDst*** pointer to a buffer for the treated message
↔ ∗***pDstSize*** size of the provide buffer / size of the treated message

Return values:

TRDP_NO_ERR no error
TRDP_MEM_ERR provided buffer to small
TRDP_COMID_ERR comid not existing

5.18.3.3 typedef void(∗ TRDP_MD_CALLBACK_T)(void ∗pRefCon, const TRDP_MD_INFO_T ∗pMsg, UINT8 ∗pData, UINT32 dataSize)

Callback for receiving indications, timeouts, releases, responses.

Parameters:

← ∗***pRefCon*** pointer to user context
← ∗***pMsg*** pointer to received message information
← ∗***pData*** pointer to received data
← ***dataSize*** size of received data pointer to received data excl. padding and FCS !!!!

5.18.3.4 typedef void(∗ TRDP_PD_CALLBACK_T)(void ∗pRefCon, const TRDP_PD_INFO_T ∗pMsg, UINT8 ∗pData, UINT32 dataSize)

Callback for receiving indications, timeouts, releases, responses.

Parameters:

← ∗***pRefCon*** pointer to user context
← ∗***pMsg*** pointer to received message information
← ∗***pData*** pointer to received data
← ***dataSize*** size of received data pointer to received data excl. padding and FCS !!!!

5.18.3.5 typedef VOS_PRINT_DBG_T TRDP_PRINT_DBG_T

TRDP configuration type definitions.

Callback function definition for error/debug output, reuse of the VOS defined function.

5.18.3.6 typedef VOS_TIME_T TRDP_TIME_T

Timer value compatible with timeval / select.

Relative or absolute date, depending on usage

5.18.3.7 typedef TRDP_ERR_T(∗ TRDP_UNMARSHALL_T)(void ∗pRefCon, UINT32 comId, const UINT8 ∗pSrc, UINT8 ∗pDst, UINT32 ∗pDstSize)

Function type for unmarshalling.

The function must know about the dataset's alignment etc.

Parameters:

← ∗***pRefCon*** pointer to user context

← ***comId*** ComId to identify the structure out of a configuration

← ∗***pSrc*** pointer to received original message

← ∗***pDst*** pointer to a buffer for the treated message

↔ ∗***pDstSize*** size of the provide buffer / size of the treated message

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR provide buffer to small

TRDP_COMID_ERR comid not existing

5.18.4 Enumeration Type Documentation

5.18.4.1 enum TRDP_DATA_TYPE_T

TRDP dataset description definitions.

Dataset element definition

Enumerator:

TRDP_BOOLEAN =UINT8, 1 bit relevant (equal to zero = false, not equal to zero = true)

TRDP_CHAR8 char, can be used also as UTF8

TRDP_UTF16 Unicode UTF-16 character.

TRDP_INT8 Signed integer, 8 bit.

TRDP_INT16 Signed integer, 16 bit.

TRDP_INT32 Signed integer, 32 bit.

TRDP_INT64 Signed integer, 64 bit.

TRDP_UINT8 Unsigned integer, 8 bit.

TRDP_UINT16 Unsigned integer, 16 bit.
TRDP_UINT32 Unsigned integer, 32 bit.
TRDP_UINT64 Unsigned integer, 64 bit.
TRDP_REAL32 Floating point real, 32 bit.
TRDP_REAL64 Floating point real, 64 bit.
TRDP_STRING Zero-terminated array of CHAR8, fixed size.
TRDP_ARRAY Array.
TRDP_RECORD Record.
TRDP_TIMEDATE32 32 bit UNIX time
TRDP_TIMEDATE48 48 bit TCN time (32 bit UNIX time and 16 bit ticks)
TRDP_TIMEDATE64 32 bit UNIX time + 32 bit miliseconds

5.18.4.2 enum TRDP_ERR_T

Return codes for all API functions.

Enumerator:

TRDP_NO_ERR No error.
TRDP_PARAM_ERR Parameter missing or out of range.
TRDP_INIT_ERR Call without valid initialization.
TRDP_NOINIT_ERR Call with invalid handle.
TRDP_TIMEOUT_ERR Timout.
TRDP_NODATA_ERR Non blocking mode: no data received.
TRDP_SOCK_ERR Socket error / option not supported.
TRDP_IO_ERR Socket IO error, data can't be received/sent.
TRDP_MEM_ERR No more memory available.
TRDP_SEMA_ERR Semaphore not available.
TRDP_QUEUE_ERR Queue empty.
TRDP_QUEUE_FULL_ERR Queue full.
TRDP_MUTEX_ERR Mutex not available.
TRDP_NOSESSION_ERR No such session.
TRDP_SESSION_ABORT_ERR Session aborted.
TRDP_NOSUB_ERR No subscriber.
TRDP_NOPUB_ERR No publisher.
TRDP_NOLIST_ERR No listener.
TRDP_CRC_ERR Wrong CRC.
TRDP_TOPO_ERR Invalid topo count.
TRDP_COMID_ERR Unknown ComId.
TRDP_STATE_ERR Call in wrong state.
TRDP_UNKNOWN_ERR Unspecified error.

5.18.4.3 enum TRDP_FLAGS_T

Various flags for PD and MD packets.

Enumerator:

TRDP_FLAGS_REDUNDANT Redundant.

TRDP_FLAGS_MARSHALL Optional marshalling/unmarshalling in TRDP stack.

TRDP_FLAGS_CALLBACK Use of callback function.

TRDP_FLAGS_TCP Use TCP for message data.

5.18.4.4 enum TRDP_MSG_T

TRDP data transfer type definitions.

Message Types

Enumerator:

TRDP_MSG_PD 'Pd' PD Data (Reply)

TRDP_MSG_PR 'Pr' PD Request

TRDP_MSG_PE 'Pe' PD Error

TRDP_MSG_MN 'Mn' MD Notification (Request without reply)

TRDP_MSG_MR 'Mr' MD Request with reply

TRDP_MSG_MP 'Mp' MD Reply without confirmation

TRDP_MSG_MQ 'Mq' MD Reply with confirmation

TRDP_MSG_MC 'Mc' MD Confirm

TRDP_MSG_ME 'Me' MD Error

5.18.4.5 enum TRDP_OPTION_T

Various flags/general TRDP options for library initialization.

Enumerator:

TRDP_OPTION_BLOCK Default: Use nonblocking I/O calls, polling necessary Set: Read calls will block, use select().

TRDP_OPTION_TRAFFIC_SHAPING Use traffic shaping - distribute packet sending.

5.18.4.6 enum TRDP_RED_STATE_T

Redundancy states.

Enumerator:

TRDP_RED_FOLLOWER Redundancy follower - redundant PD will be not sent out.

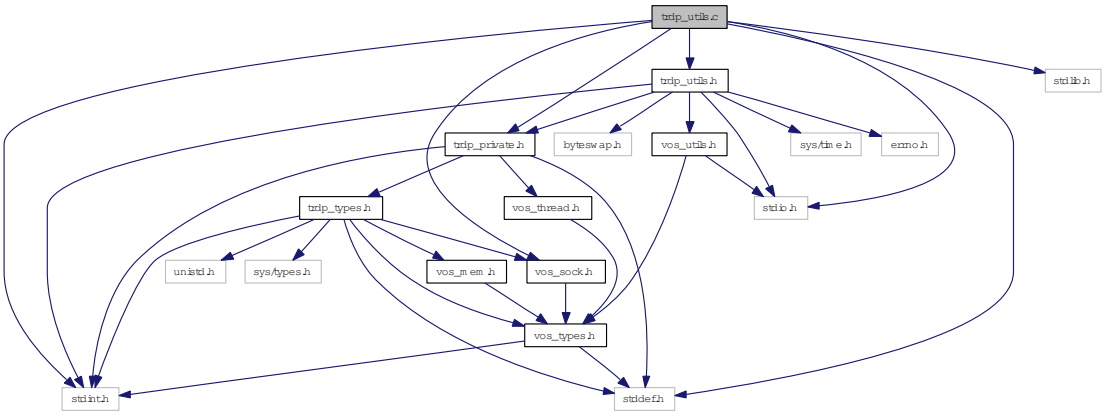
TRDP_RED_LEADER Redundancy leader - redundant PD will be sent out.

5.19 trdp_utils.c File Reference

Helper functions for TRDP communication.

```
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include "vos_sock.h"
#include "trdp_private.h"
#include "trdp_utils.h"
```

Include dependency graph for trdp_utils.c:



Functions

- int am_big_endian ()

  Determine if we are Big or Little endian.

- PD_ELE_T * trdp_util_getnext (PD_ELE_T *pHead, const struct timeval *pNow, PD_ELE_T **ppNextElement)

  Find the packet which has to be send next.

- UINT32 trdp_packetSizePD (UINT32 dataSize)

  Get the packet size from the raw data size.

- PD_ELE_T * trdp_queue_find_comId (PD_ELE_T **ppHead, uint32_t comId)

  Return the element with same comId.

- PD_ELE_T * trdp_queue_find_addr (PD_ELE_T *pHead, TRDP_ADDRESSES *addr)

  Return the element with same comId.

- void trdp_queue_del_element (PD_ELE_T **ppHead, PD_ELE_T *pDelete)

  Delete an element.

- void trdp_queue_app_last (PD_ELE_T **ppHead, PD_ELE_T *pNew)

  Append an element at end of queue.

- void trdp_queue_ins_first (PD_ELE_T **ppHead, PD_ELE_T *pNew)

  Insert an element at front of queue.

- void trdp_initSockets (TRDP_SOCKETS_T iface[ ])

  Handle the socket pool: Initialize it.

- TRDP_ERR_T trdp_requestSocket (TRDP_SOCKETS_T iface[ ], const TRDP_SEND_PARAM_-T *params, TRDP_IP_ADDR_T srcIP, TRDP_SOCK_TYPE_T usage, TRDP_OPTION_T options, INT32 *pIndex)

  Handle the socket pool: Request a socket from our socket pool.

- TRDP_ERR_T trdp_releaseSocket (TRDP_SOCKETS_T iface[ ], INT32 index)

  Handle the socket pool: Release a socket from our socket pool.

5.19.1 Detailed Description

Helper functions for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_utils.c 2 2012-06-04 11:25:16Z 97025

5.19.2 Function Documentation

5.19.2.1 int am_big_endian ()

Determine if we are Big or Little endian.

Return values:

!= 0 we are big endian

0 we are little endian

5.19.2.2 void trdp_initSockets (TRDP_SOCKETS_T *iface*[ ])

Handle the socket pool: Initialize it.

Parameters:

← ***iface*** pointer to the socket pool

5.19.2.3 UINT32 trdp_packetSizePD (UINT32 *dataSize*)

Get the packet size from the raw data size.

Parameters:

← ***dataSize*** net data size (without padding or FCS)

Return values:

packet size the size of the complete packet to be sent or received

5.19.2.4 void trdp_queue_app_last (PD_ELE_T ** *ppHead*, PD_ELE_T * *pNew*)

Append an element at end of queue.

Parameters:

← ***ppHead*** pointer to pointer to head of queue
← ***pNew*** pointer to element to append

5.19.2.5 void trdp_queue_del_element (PD_ELE_T ** *ppHead*, PD_ELE_T * *pDelete*)

Delete an element.

Parameters:

← ***ppHead*** pointer to pointer to head of queue
← ***pDelete*** pointer to element to delete

5.19.2.6 PD_ELE_T* trdp_queue_find_addr (PD_ELE_T * *pHead*, TRDP_ADDRESSES * *addr*)

Return the element with same comId.

Parameters:

← ***pHead*** pointer to head of queue
← ***addr*** Pub/Sub handle (Address, ComID, srcIP & dest IP) to search for

Return values:

!= NULL pointer to PD element
NULL No PD element found

5.19.2.7 PD_ELE_T* trdp_queue_find_comId (PD_ELE_T ** *ppHead*, uint32_t *comId*)

Return the element with same comId.

Parameters:

← ***ppHead*** pointer to pointer to head of queue
← ***comId*** ComID to search for

Return values:

!= NULL pointer to PD element
NULL No PD element found

5.19.2.8 void trdp_queue_ins_first (PD_ELE_T ** *ppHead*, PD_ELE_T * *pNew*)

Insert an element at front of queue.

Parameters:

← ***ppHead*** pointer to pointer to head of queue
← ***pNew*** pointer to element to insert

5.19.2.9 TRDP_ERR_T trdp_releaseSocket (TRDP_SOCKETS_T *iface*[ ], INT32 *index*)

Handle the socket pool: Release a socket from our socket pool.

Parameters:

↔ ***iface*** socket pool
← ***index*** index of socket to release

Return values:

TRDP_NO_ERR
TRDP_PARAM_ERR

Here is the call graph for this function:

5.19.2.10 TRDP_ERR_T trdp_requestSocket (TRDP_SOCKETS_T *iface*[ ], const TRDP_SEND_PARAM_T * *params*, TRDP_IP_ADDR_T *srcIP*, TRDP_SOCK_TYPE_T *usage*, TRDP_OPTION_T *options*, INT32 * *pIndex*)

Handle the socket pool: Request a socket from our socket pool.

Parameters:

↔ ***iface*** socket pool

← ***params*** parameters to use

← ***srcIP*** IP to bind to (0 = any address)

← ***usage*** type and port to bind to

← ***options*** blocking/nonblocking

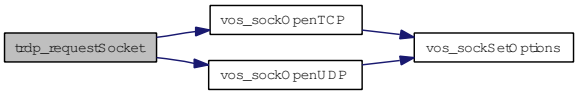
→ ***pIndex*** returned index of socket pool

Return values:

TRDP_NO_ERR

TRDP_PARAM_ERR

Here is the call graph for this function:



5.19.2.11 PD_ELE_T* trdp_util_getnext (PD_ELE_T * *pHead*, const struct timeval * *pNow*, PD_ELE_T ** *ppNextElement*)

Find the packet which has to be send next.

Parameters:

← ***pHead*** pointer to first queue element

← ***pNow*** Current time

→ ***ppNextElement*** pointer to pointer to PD element

Return values:

!= NULL pointer to PD packet

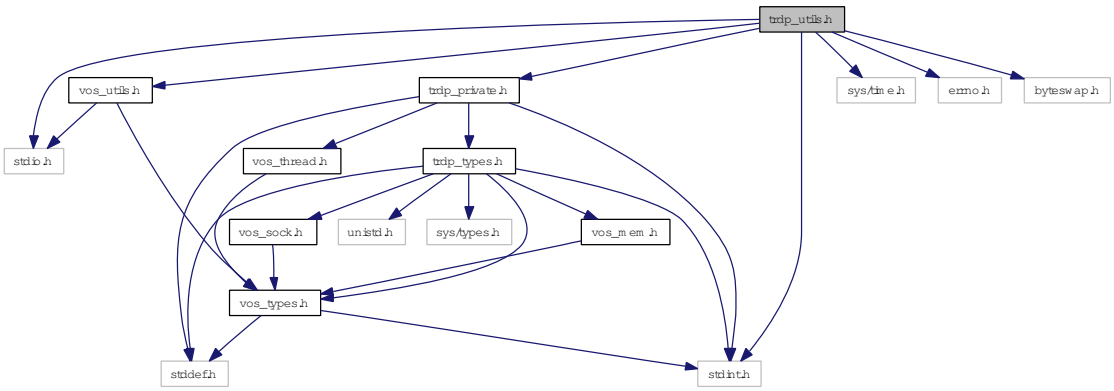
NULL No PD packet found

5.20 trdp_utils.h File Reference

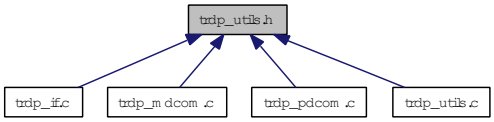
Common utilities for TRDP communication.

```
#include <stdio.h>
#include <stdint.h>
#include <sys/time.h>
#include <errno.h>
#include <byteswap.h>
#include "trdp_private.h"
#include "vos_utils.h"
```

Include dependency graph for trdp_utils.h:



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



Functions

- int am_big_endian ()

  Determine if we are Big or Little endian.

- PD_ELE_T * trdp_util_getnext (PD_ELE_T *pHead, const struct timeval *pNow, PD_ELE_T **ppEle)

  Find the packet which has to be send next.

- PD_ELE_T * trdp_queue_find_addr (PD_ELE_T *pHead, TRDP_ADDRESSES *pAddr)

  Return the element with same comId.

- void trdp_queue_del_element (PD_ELE_T **pHead, PD_ELE_T *pDelete)

Delete an element.

- void trdp_queue_app_last (PD_ELE_T **pHead, PD_ELE_T *pNew)
  Append an element at end of queue.
- void trdp_queue_ins_first (PD_ELE_T **pHead, PD_ELE_T *pNew)
  Insert an element at front of queue.
- void trdp_initSockets (TRDP_SOCKETS_T iface[ ])
  Handle the socket pool: Initialize it.
- TRDP_ERR_T trdp_requestSocket (TRDP_SOCKETS_T iface[ ], const TRDP_SEND_PARAM_T *params, TRDP_IP_ADDR_T srcIP, TRDP_SOCK_TYPE_T usage, TRDP_OPTION_T options, INT32 *pIndex)
  Handle the socket pool: Request a socket from our socket pool.
- TRDP_ERR_T trdp_releaseSocket (TRDP_SOCKETS_T iface[ ], INT32 index)
  Handle the socket pool: Release a socket from our socket pool.
- UINT32 trdp_packetSizePD (UINT32 dataSize)
  Get the packet size from the raw data size.

5.20.1 Detailed Description

Common utilities for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

trdp_utils.h 2 2012-06-04 11:25:16Z 97025

5.20.2 Function Documentation

5.20.2.1 int am_big_endian ()

Determine if we are Big or Little endian.

Return values:

!= 0 we are big endian

0 we are little endian

5.20.2.2 void trdp_initSockets (TRDP_SOCKETS_T *iface*[ ])

Handle the socket pool: Initialize it.

Parameters:

← ***iface*** pointer to the socket pool

5.20.2.3 UINT32 trdp_packetSizePD (UINT32 *dataSize*)

Get the packet size from the raw data size.

Parameters:

← ***dataSize*** net data size (without padding or FCS)

Return values:

packet size the size of the complete packet to be sent or received

5.20.2.4 void trdp_queue_app_last (PD_ELE_T ** *ppHead*, PD_ELE_T * *pNew*)

Append an element at end of queue.

Parameters:

← ***ppHead*** pointer to pointer to head of queue
← ***pNew*** pointer to element to append

5.20.2.5 void trdp_queue_del_element (PD_ELE_T ** *ppHead*, PD_ELE_T * *pDelete*)

Delete an element.

Parameters:

← ***ppHead*** pointer to pointer to head of queue
← ***pDelete*** pointer to element to delete

5.20.2.6 PD_ELE_T* trdp_queue_find_addr (PD_ELE_T * *pHead*, TRDP_ADDRESSES * *addr*)

Return the element with same comId.

Parameters:

← ***pHead*** pointer to head of queue
← ***addr*** Pub/Sub handle (Address, ComID, srcIP & dest IP) to search for

Return values:

!= NULL pointer to PD element
NULL No PD element found

5.20.2.7 void trdp_queue_ins_first (PD_ELE_T ** *ppHead*, PD_ELE_T * *pNew*)

Insert an element at front of queue.

Parameters:

← ***ppHead*** pointer to pointer to head of queue

← ***pNew*** pointer to element to insert

5.20.2.8 TRDP_ERR_T trdp_releaseSocket (TRDP_SOCKETS_T *iface*[ ], INT32 *index*)

Handle the socket pool: Release a socket from our socket pool.

Parameters:

↔ ***iface*** socket pool

← ***index*** index of socket to release

Return values:

TRDP_NO_ERR

TRDP_PARAM_ERR

Here is the call graph for this function:

5.20.2.9 TRDP_ERR_T trdp_requestSocket (TRDP_SOCKETS_T *iface*[ ], const TRDP_SEND_PARAM_T * *params*, TRDP_IP_ADDR_T *srcIP*, TRDP_SOCK_TYPE_T *usage*, TRDP_OPTION_T *options*, INT32 * *pIndex*)

Handle the socket pool: Request a socket from our socket pool.

Parameters:

↔ ***iface*** socket pool

← ***params*** parameters to use

← ***srcIP*** IP to bind to (0 = any address)

← ***usage*** type and port to bind to

← ***options*** blocking/nonblocking

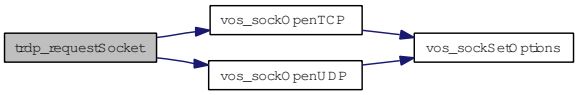
→ ***pIndex*** returned index of socket pool

Return values:

TRDP_NO_ERR

TRDP_PARAM_ERR

Here is the call graph for this function:



5.20.2.10 PD_ELE_T* trdp_util_getnext (PD_ELE_T * *pHead*, const struct timeval * *pNow*, PD_ELE_T ** *ppNextElement*)

Find the packet which has to be send next.

Parameters:

← ***pHead*** pointer to first queue element

← ***pNow*** Current time

→ ***ppNextElement*** pointer to pointer to PD element

Return values:

!= NULL pointer to PD packet

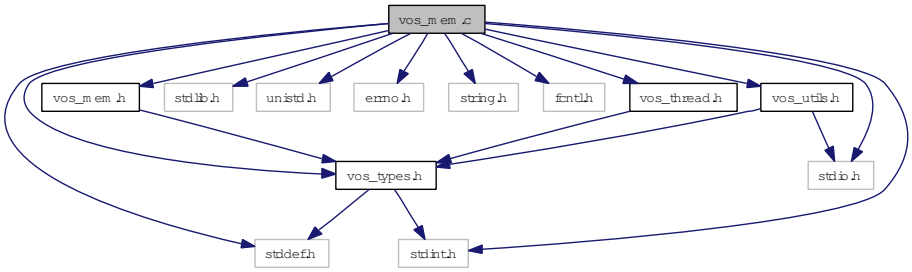
NULL No PD packet found

5.21 vos_mem.c File Reference

Memory functions.

```
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include "vos_types.h"
#include "vos_utils.h"
#include "vos_mem.h"
#include "vos_thread.h"
```

Include dependency graph for vos_mem.c:



Functions

- EXT_DECL VOS_ERR_T vos_memInit (UINT8 *pMemoryArea, UINT32 size, const UINT32 fragMem[VOS_MEM_NBLOCKSIZES])

  Initialize the memory unit.

- EXT_DECL VOS_ERR_T vos_memDelete (UINT8 *pMemoryArea)

  Delete the memory area.

- EXT_DECL UINT8 * vos_memAlloc (UINT32 size)

  Allocate a block of memory (from memory area above).

- EXT_DECL VOS_ERR_T vos_memFree (void *pMemBlock)

  Deallocate a block of memory (from memory area above).

- EXT_DECL VOS_ERR_T vos_memCount (UINT32 *pAllocatedMemory, UINT32 *pFreeMemory, UINT32 *pFragMem[VOS_MEM_NBLOCKSIZES])

  Return used and available memory (of memory area above).

- EXT_DECL VOS_ERR_T vos_queueCreate (const CHAR8 *pKey, VOS_QUEUE_T *pQueueID, UINT32 maxNoMsg, UINT32 maxLength)

  Initialize a message queue.

- EXT_DECL VOS_ERR_T vos_queueDestroy (VOS_QUEUE_T queueID)

  Destroy a message queue.

- EXT_DECL VOS_ERR_T vos_queueSend (VOS_QUEUE_T queueID, const UINT8 *pMsg, UINT32 size)

  Send a message.

- EXT_DECL VOS_ERR_T vos_queueReceive (VOS_QUEUE_T queueID, UINT8 *pMsg, UINT32 *pSize, UINT32 usTimeout)

  Get a message.

- EXT_DECL VOS_ERR_T vos_sharedOpen (const CHAR8 *pKey, VOS_SHRD_T *pHandle, UINT8 **ppMemoryArea, UINT32 *pSize)

  Create a shared memory area or attach to existing one.

- EXT_DECL VOS_ERR_T vos_sharedClose (VOS_SHRD_T handle, const UINT8 *pMemoryArea)

  Close connection to the shared memory area.

5.21.1 Detailed Description

Memory functions.

OS abstraction of memory access and control

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

vos_mem.c 2 2012-06-04 11:25:16Z 97025

5.21.2 Function Documentation

5.21.2.1 EXT_DECL UINT8* vos_memAlloc (UINT32 *size*)

Allocate a block of memory (from memory area above).

Parameters:

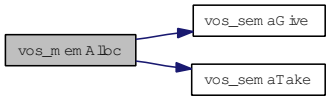
← ***size*** Size of requested block

Return values:

Pointer to memory area

NULL if no memory available

Here is the call graph for this function:



5.21.2.2 EXT_DECL VOS_ERR_T vos_memCount (UINT32 * *pAllocatedMemory*, UINT32 * *pFreeMemory*, UINT32 * *pFragMem*[VOS_MEM_NBLOCKSIZES])

Return used and available memory (of memory area above).

Parameters:

→ ***pAllocatedMemory*** Pointer to allocated memory size

→ ***pFreeMemory*** Pointer to free memory size

→ ***pFragMem*** Pointer to list of used memoryblocks

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

5.21.2.3 EXT_DECL VOS_ERR_T vos_memDelete (UINT8 * *pMemoryArea*)

Delete the memory area.

This will eventually invalidate any previously allocated memory blocks! It should be called last before the application quits. No further access to the memory blocks is allowed after this call.

Parameters:

← ***pMemoryArea*** Pointer to memory area to use

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

5.21.2.4 EXT_DECL VOS_ERR_T vos_memFree (void * *pMemBlock*)

Deallocate a block of memory (from memory area above).

Parameters:

← ***pMemBlock*** Pointer to memory block to be freed

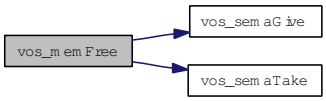
Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

Here is the call graph for this function:



5.21.2.5 EXT_DECL VOS_ERR_T vos_memInit (UINT8 * *pMemoryArea*, UINT32 *size*, const UINT32 *fragMem*[VOS_MEM_NBLOCKSIZES])

Initialize the memory unit.

Init a supplied block of memory and prepare it for use with vos_alloc and vos_dealloc. The used block sizes can be supplied and will be preallocated.

Parameters:

← ***pMemoryArea*** Pointer to memory area to use

← ***size*** Size of provided memory area

← ***fragMem*** Pointer to list of preallocated block sizes, used to fragment memory for large blocks

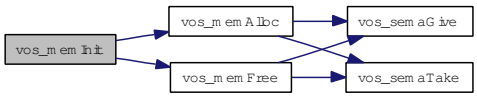
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

VOS_MEM_ERR no memory available

Here is the call graph for this function:



5.21.2.6 EXT_DECL VOS_ERR_T vos_queueCreate (const CHAR8 * *pKey*, VOS_QUEUE_T * *pQueueID*, UINT32 *maxNoMsg*, UINT32 *maxLength*)

Initialize a message queue.

Returns a handle for further calls

Parameters:

← ***pKey*** Unique identifier (file name)

→ ***pQueueID*** Pointer to returned queue handle

← ***maxNoMsg*** maximum number of messages

← ***maxLength*** maximum size of one messages

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_INIT_ERR not supported

VOS_QUEUE_ERR error creating queue

5.21.2.7 EXT_DECL VOS_ERR_T vos_queueDestroy (VOS_QUEUE_T *queueID*)

Destroy a message queue.

Free all resources used by this queue

Parameters:

← ***queueID*** Queue handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

5.21.2.8 EXT_DECL VOS_ERR_T vos_queueReceive (VOS_QUEUE_T *queueID*, UINT8 * *pMsg*, UINT32 * *pSize*, UINT32 *usTimeout*)

Get a message.

Parameters:

← ***queueID*** Queue handle

→ ***pMsg*** Pointer to message to be received

↔ ***pSize*** Pointer to max. message size on entry, actual size on exit

← ***usTimeout*** Maximum time to wait for a message in usec

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_QUEUE_ERR queue is empty

5.21.2.9 EXT_DECL VOS_ERR_T vos_queueSend (VOS_QUEUE_T *queueID*, const UINT8 * *pMsg*, UINT32 *size*)

Send a message.

Parameters:

← ***queueID*** Queue handle

← ***pMsg*** Pointer to message to be sent

← ***size*** Message size

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_QUEUE_FULL queue is full

5.21.2.10 EXT_DECL VOS_ERR_T vos_sharedClose (VOS_SHRD_T *handle*, const UINT8 * *pMemoryArea*)

Close connection to the shared memory area.

If the area was created by the calling process, the area will be closed (freed). If the area was attached, it will be detached. This function is not available in each target implementation.

Parameters:

← ***handle*** Returned handle

← ***pMemoryArea*** Pointer to memory area

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

5.21.2.11 EXT_DECL VOS_ERR_T vos_sharedOpen (const CHAR8 * *pKey*, VOS_SHRD_T * *pHandle*, UINT8 ** *ppMemoryArea*, UINT32 * *pSize*)

Create a shared memory area or attach to existing one.

The first call with the a specified key will create a shared memory area with the supplied size and will return a handle and a pointer to that area. If the area already exists, the area will be attached. This function is not available in each target implementation.

Parameters:

← ***pKey*** Unique identifier (file name)

→ ***pHandle*** Pointer to returned handle

→ ***ppMemoryArea*** Pointer to pointer to memory area

↔ ***pSize*** Pointer to size of area to allocate, on return actual size after attach

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

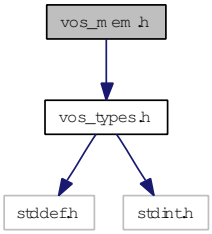
VOS_MEM_ERR no memory available

5.22 vos_mem.h File Reference

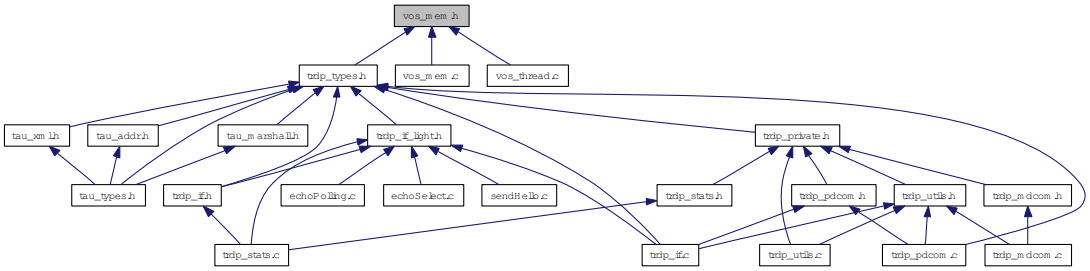
Memory and queue functions for OS abstraction.

```
#include "vos_types.h"
```

Include dependency graph for vos_mem.h:



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



Defines

- #define VOS_MEM_BLOCKSIZES

  We internally allocate memory always by these block sizes.

- #define VOS_MEM_PREALLOCATE {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 4, 0, 0}

  Default pre-allocation of free memory blocks.

Typedefs

- typedef struct VOS_QUEUE_T * VOS_QUEUE_T

  Opaque queue define.

Enumerations

- enum VOS_MEM_BLK_T

  enumeration for memory block sizes

Functions

- EXT_DECL VOS_ERR_T vos_memInit (UINT8 *pMemoryArea, UINT32 size, const UINT32 fragMem[VOS_MEM_NBLOCKSIZES])

  Initialize the memory unit.

- EXT_DECL VOS_ERR_T vos_memDelete (UINT8 *pMemoryArea)

  Delete the memory area.

- EXT_DECL UINT8 * vos_memAlloc (UINT32 size)

  Allocate a block of memory (from memory area above).

- EXT_DECL VOS_ERR_T vos_memFree (void *pMemBlock)

  Deallocate a block of memory (from memory area above).

- EXT_DECL VOS_ERR_T vos_memCount (UINT32 *pAllocatedMemory, UINT32 *pFreeMemory, UINT32 *pFragMem[VOS_MEM_NBLOCKSIZES])

  Return used and available memory (of memory area above).

- EXT_DECL VOS_ERR_T vos_queueCreate (const CHAR8 *pKey, VOS_QUEUE_T *pQueueId, UINT32 maxNoMsg, UINT32 maxLength)

  Initialize a message queue.

- EXT_DECL VOS_ERR_T vos_queueDestroy (VOS_QUEUE_T queueID)

  Destroy a message queue.

- EXT_DECL VOS_ERR_T vos_queueSend (VOS_QUEUE_T queueID, const UINT8 *pMsg, UINT32 size)

  Send a message.

- EXT_DECL VOS_ERR_T vos_queueReceive (VOS_QUEUE_T queueID, UINT8 *pMsg, UINT32 *pSize, UINT32 usTimeout)

  Get a message.

- EXT_DECL VOS_ERR_T vos_sharedOpen (const CHAR8 *pKey, VOS_SHRD_T *pHandle, UINT8 **ppMemoryArea, UINT32 *pSize)

  Create a shared memory area or attach to existing one.

- EXT_DECL VOS_ERR_T vos_sharedClose (VOS_SHRD_T handle, const UINT8 *pMemoryArea)

  Close connection to the shared memory area.

5.22.1 Detailed Description

Memory and queue functions for OS abstraction.

This module provides three services: 1. A memory control supervison

- Private memory management with optimised fragmentation handling

- A message queue handler (system-wide on supported systems)
- Access to shared memory (on supported systems only) Within the prototype TRDP stack, only the memory management unit is currently in use.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH Peter Brander (Memory scheme)

Remarks:

Id

vos_mem.h 2 2012-06-04 11:25:16Z 97025

5.22.2 Define Documentation

5.22.2.1 #define VOS_MEM_BLOCKSIZES

Value:

```
{32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, \
                            16384, 32768, 65536, 131072, 262144, 524288}
```

We internally allocate memory always by these block sizes.

The largest available block is 524288 Bytes, provided the overal size of the used memory allocation area is larger.

5.22.2.2 #define VOS_MEM_PREALLOCATE {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 4, 0, 0}

Default pre-allocation of free memory blocks.

To avoid problems with too many small blocks and no large one. Specify how many of each block size that should be pre-allocated (and freed!) to pre-segment the memory area.

5.22.3 Function Documentation

5.22.3.1 EXT_DECL UINT8* vos_memAlloc (UINT32 *size*)

Allocate a block of memory (from memory area above).

Parameters:

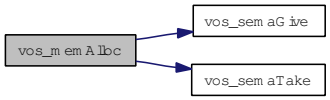
← ***size*** Size of requested block

Return values:

Pointer to memory area

NULL if no memory available

Here is the call graph for this function:



5.22.3.2 EXT_DECL VOS_ERR_T vos_memCount (UINT32 * *pAllocatedMemory*, UINT32 * *pFreeMemory*, UINT32 * *pFragMem*[VOS_MEM_NBLOCKSIZES])

Return used and available memory (of memory area above).

Parameters:

→ ***pAllocatedMemory*** Pointer to allocated memory size

→ ***pFreeMemory*** Pointer to free memory size

→ ***pFragMem*** Pointer to list of used memoryblocks

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

5.22.3.3 EXT_DECL VOS_ERR_T vos_memDelete (UINT8 * *pMemoryArea*)

Delete the memory area.

This will eventually invalidate any previously allocated memory blocks! It should be called last before the application quits. No further access to the memory blocks is allowed after this call.

Parameters:

← ***pMemoryArea*** Pointer to memory area to use

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

5.22.3.4 EXT_DECL VOS_ERR_T vos_memFree (void * *pMemBlock*)

Deallocate a block of memory (from memory area above).

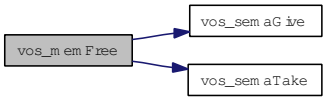
Parameters:

← ***pMemBlock*** Pointer to memory block to be freed

Return values:

VOS_NO_ERR no error
VOS_INIT_ERR module not initialised
VOS_PARAM_ERR parameter out of range/invalid

Here is the call graph for this function:



5.22.3.5 EXT_DECL VOS_ERR_T vos_memInit (UINT8 * *pMemoryArea*, UINT32 *size*, const UINT32 *fragMem*[VOS_MEM_NBLOCKSIZES])

Initialize the memory unit.

Init a supplied block of memory and prepare it for use with vos_alloc and vos_dealloc. The used block sizes can be supplied and will be preallocated.

Parameters:

← ***pMemoryArea*** Pointer to memory area to use
← ***size*** Size of provided memory area
← ***fragMem*** Pointer to list of preallocate block sizes, used to fragment memory for large blocks

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR parameter out of range/invalid
VOS_MEM_ERR no memory available

Init a supplied block of memory and prepare it for use with vos_alloc and vos_dealloc. The used block sizes can be supplied and will be preallocated.

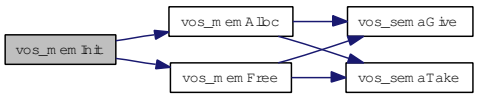
Parameters:

← ***pMemoryArea*** Pointer to memory area to use
← ***size*** Size of provided memory area
← ***fragMem*** Pointer to list of preallocated block sizes, used to fragment memory for large blocks

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR parameter out of range/invalid
VOS_MEM_ERR no memory available

Here is the call graph for this function:



5.22.3.6 EXT_DECL VOS_ERR_T vos_queueCreate (const CHAR8 * *pKey*, VOS_QUEUE_T * *pQueueID*, UINT32 *maxNoMsg*, UINT32 *maxLength*)

Initialize a message queue.

Returns a handle for further calls

Parameters:

← ***pKey*** Unique identifier (file name)

→ ***pQueueID*** Pointer to returned queue handle

← ***maxNoMsg*** maximum number of messages

← ***maxLength*** maximum size of one messages

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_INIT_ERR not supported

VOS_QUEUE_ERR error creating queue

5.22.3.7 EXT_DECL VOS_ERR_T vos_queueDestroy (VOS_QUEUE_T *queueID*)

Destroy a message queue.

Free all resources used by this queue

Parameters:

← ***queueID*** Queue handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

5.22.3.8 EXT_DECL VOS_ERR_T vos_queueReceive (VOS_QUEUE_T *queueID*, UINT8 * *pMsg*, UINT32 * *pSize*, UINT32 *usTimeout*)

Get a message.

Parameters:

← ***queueID*** Queue handle

→ ***pMsg*** Pointer to message to be received

↔ ***pSize*** Pointer to max. message size on entry, actual size on exit

← ***usTimeout*** Maximum time to wait for a message in usec

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_QUEUE_ERR queue is empty

5.22.3.9 EXT_DECL VOS_ERR_T vos_queueSend (VOS_QUEUE_T *queueID*, const UINT8 * *pMsg*, UINT32 *size*)

Send a message.

Parameters:

← ***queueID*** Queue handle

← ***pMsg*** Pointer to message to be sent

← ***size*** Message size

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_QUEUE_FULL queue is full

5.22.3.10 EXT_DECL VOS_ERR_T vos_sharedClose (VOS_SHRD_T *handle*, const UINT8 * *pMemoryArea*)

Close connection to the shared memory area.

If the area was created by the calling process, the area will be closed (freed). If the area was attached, it will be detached. This function is not available in each target implementation.

Parameters:

← ***handle*** Returned handle

← ***pMemoryArea*** Pointer to memory area

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

5.22.3.11 EXT_DECL VOS_ERR_T vos_sharedOpen (const CHAR8 * *pKey*, VOS_SHRD_T * *pHandle*, UINT8 ** *ppMemoryArea*, UINT32 * *pSize*)

Create a shared memory area or attach to existing one.

The first call with the a specified key will create a shared memory area with the supplied size and will return a handle and a pointer to that area. If the area already exists, the area will be attached. This function is not available in each target implementation.

Parameters:

← ***pKey*** Unique identifier (file name)

→ ***pHandle*** Pointer to returned handle

→ ***ppMemoryArea*** Pointer to pointer to memory area

↔ ***pSize*** Pointer to size of area to allocate, on return actual size after attach

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

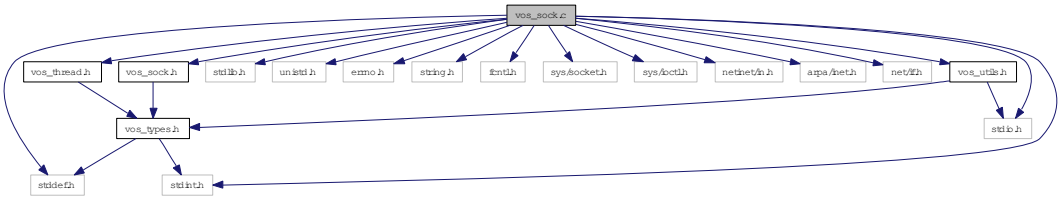
VOS_MEM_ERR no memory available

5.23 vos_sock.c File Reference

Socket functions.

```
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <net/if.h>
#include "vos_utils.h"
#include "vos_sock.h"
#include "vos_thread.h"
```

Include dependency graph for vos_sock.c:



Functions

- EXT_DECL UINT16 vos_htons (UINT16 val)

  Byte swapping.

- EXT_DECL UINT16 vos_ntohs (UINT16 val)

  Byte swapping 2 Bytes.

- EXT_DECL UINT32 vos_htonl (UINT32 val)

  Byte swapping 4 Bytes.

- EXT_DECL UINT32 vos_ntohl (UINT32 val)

  Byte swapping 4 Bytes.

- EXT_DECL BOOL vos_IsMulticast (UINT32 ipAddress)

  Check if the supplied address is a multicast group address.

- EXT_DECL VOS_ERR_T vos_sockInit (void)

  Initialize the socket library.

- EXT_DECL VOS_ERR_T vos_sockOpenUDP (INT32 *pSock, const VOS_SOCK_OPT_T *pOptions)

  Create an UDP socket.

- EXT_DECL VOS_ERR_T vos_sockOpenTCP (INT32 *pSock, const VOS_SOCK_OPT_T *pOptions)

  Create a TCP socket.

- EXT_DECL VOS_ERR_T vos_sockClose (INT32 sock)

  Close a socket.

- EXT_DECL VOS_ERR_T vos_sockSetOptions (INT32 sock, const VOS_SOCK_OPT_T *pOptions)

  Set socket options.

- EXT_DECL VOS_ERR_T vos_sockJoinMC (INT32 sock, UINT32 mcAddress, UINT32 ipAddress)

  Join a multicast group.

- EXT_DECL VOS_ERR_T vos_sockLeaveMC (INT32 sock, UINT32 mcAddress, UINT32 ipAddress)

  Leave a multicast group.

- EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 sock, const UINT8 *pBuffer, UINT32 size, UINT32 ipAddress, UINT16 port)

  Send UDP data.

- EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 sock, UINT8 *pBuffer, INT32 *pSize, UINT32 *pIPAddr)

  Receive UDP data.

- EXT_DECL VOS_ERR_T vos_sockBind (INT32 sock, UINT32 ipAddress, UINT16 port)

  Bind a socket to an address and port.

- EXT_DECL VOS_ERR_T vos_sockListen (INT32 sock, UINT32 backlog)

  Listen for incoming connections.

- EXT_DECL VOS_ERR_T vos_sockAccept (INT32 sock, INT32 *pSock, UINT32 *pIPAddress, UINT16 *pPort)

  Accept an incoming TCP connection.

- EXT_DECL VOS_ERR_T vos_sockConnect (INT32 sock, UINT32 ipAddress, UINT16 port)

  Open a TCP connection.

- EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 sock, const UINT8 *pBuffer, UINT32 size)
  Send TCP data.
- EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 sock, UINT8 *pBuffer, INT32 *pSize)
  Receive TCP data.

5.23.1 Detailed Description

Socket functions.

OS abstraction of IP socket functions for UDP and TCP

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

vos_sock.c 2 2012-06-04 11:25:16Z 97025

5.23.2 Function Documentation

5.23.2.1 EXT_DECL UINT32 vos_htonl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

5.23.2.2 EXT_DECL UINT16 vos_htons (UINT16 *val*)

Byte swapping.

Byte swapping 2 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

5.23.2.3 EXT_DECL BOOL vos_IsMulticast (UINT32 *ipAddress*)

Check if the supplied address is a multicast group address.

Parameters:

← ***ipAddress*** IP address to check.

Return values:

TRUE address is multicast

FALSE address is not a multicast address

5.23.2.4 EXT_DECL UINT32 vos_ntohl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

5.23.2.5 EXT_DECL UINT16 vos_ntohs (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

5.23.2.6 EXT_DECL VOS_ERR_T vos_sockAccept (INT32 *sock*, INT32 * *pSock*, UINT32 * *pIPAddress*, UINT16 * *pPort*)

Accept an incoming TCP connection.

Accept incoming connections on the provided socket. May block and will return a new socket descriptor when accepting a connection. The original socket *pSock, remains open.

Parameters:

← ***sock*** Socket descriptor

→ ***pSock*** Pointer to socket descriptor, on exit new socket

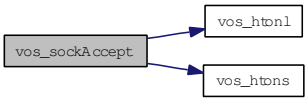
→ ***pIPAddress*** source IP to receive on, 0 for any

→ ***pPort*** port to receive on, 20548 for PD

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR NULL parameter, parameter error
VOS_UNKNOWN_ERR sock descriptor unknown error

Here is the call graph for this function:



5.23.2.7 EXT_DECL VOS_ERR_T vos_sockBind (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Bind a socket to an address and port.

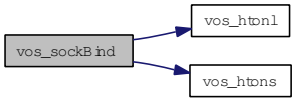
Parameters:

← ***sock*** socket descriptor
← ***ipAddress*** source IP to receive on, 0 for any
← ***port*** port to receive on, 20548 for PD

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR sock descriptor unknown, parameter error
VOS_IO_ERR Input/Output error
VOS_MEM_ERR resource error

Here is the call graph for this function:



5.23.2.8 EXT_DECL VOS_ERR_T vos_sockClose (INT32 *sock*)

Close a socket.

Release any resources aquired by this socket

Parameters:

← ***sock*** socket descriptor

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR sock descriptor unknown

5.23.2.9 EXT_DECL VOS_ERR_T vos_sockConnect (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Open a TCP connection.

Parameters:

← ***sock*** socket descriptor

← ***ipAddress*** destination IP

← ***port*** destination port

Return values:

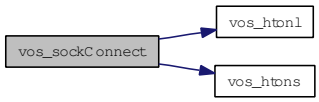
VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

Here is the call graph for this function:



5.23.2.10 EXT_DECL VOS_ERR_T vos_sockInit (void)

Initialize the socket library.

Must be called once before any other call

Return values:

VOS_NO_ERR no error

VOS_SOCK_ERR sockets not supported

5.23.2.11 EXT_DECL VOS_ERR_T vos_sockJoinMC (INT32 *sock*, UINT32 *mcAddress*, UINT32 *ipAddress*)

Join a multicast group.

Note: Some targeted systems might not support this option.

Parameters:

← ***sock*** socket descriptor

← ***mcAddress*** multicast group to join

← ***ipAddress*** depicts interface on which to join, default 0 for any

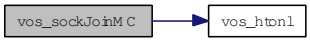
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_SOCK_ERR option not supported

Here is the call graph for this function:



5.23.2.12 EXT_DECL VOS_ERR_T vos_sockLeaveMC (INT32 *sock*, UINT32 *mcAddress*, UINT32 *ipAddress*)

Leave a multicast group.

Note: Some targeted systems might not support this option.

Parameters:

← ***sock*** socket descriptor

← ***mcAddress*** multicast group to join

← ***ipAddress*** depicts interface on which to leave, default 0 for any

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_SOCK_ERR option not supported

Here is the call graph for this function:

5.23.2.13 EXT_DECL VOS_ERR_T vos_sockListen (INT32 *sock*, UINT32 *backlog*)

Listen for incoming connections.

Listen for incoming TCP connections.

Parameters:

← ***sock*** socket descriptor

← ***backlog*** maximum connection attempts if system is busy

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

5.23.2.14 EXT_DECL VOS_ERR_T vos_sockOpenTCP (INT32 * *pSock*, const VOS_SOCK_OPT_T * *pOptions*)

Create a TCP socket.

Return a socket descriptor for further calls. The socket options are optional and can be applied later.

Parameters:

→ ***pSock*** pointer to socket descriptor returned

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pSock == NULL

VOS_SOCK_ERR socket not available or option not supported

Here is the call graph for this function:

5.23.2.15 EXT_DECL VOS_ERR_T vos_sockOpenUDP (INT32 * *pSock*, const VOS_SOCK_OPT_T * *pOptions*)

Create an UDP socket.

Return a socket descriptor for further calls. The socket options are optional and can be applied later. Note: Some targeted systems might not support every option.

Parameters:

→ ***pSock*** pointer to socket descriptor returned

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pSock == NULL

VOS_SOCK_ERR socket not available or option not supported

Here is the call graph for this function:

5.23.2.16 EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 *sock*, UINT8 * *pBuffer*, INT32 * *pSize*)

Receive TCP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error occured. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned.

Parameters:

← ***sock*** socket descriptor

→ ***pBuffer*** pointer to applications data buffer

↔ ***pSize*** pointer to the received data size

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be read

VOS_NODATA_ERR no data in non-blocking

5.23.2.17 EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 *sock*, UINT8 * *pBuffer*, INT32 * *pSize*, UINT32 * *pIPAddr*)

Receive UDP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error occured. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned.

Parameters:

← ***sock*** socket descriptor

→ ***pBuffer*** pointer to applications data buffer

↔ ***pSize*** pointer to the received data size

→ ***pIPAddr*** source IP

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be read

VOS_NODATA_ERR no data in non-blocking

Here is the call graph for this function:

5.23.2.18 EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 *size*)

Send TCP data.

Send data to the supplied address and port.

Parameters:

← ***sock*** socket descriptor

← ***pBuffer*** pointer to data to send

← ***size*** size of the data to send

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be sent

5.23.2.19 EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 *size*, UINT32 *ipAddress*, UINT16 *port*)

Send UDP data.

Send data to the supplied address and port.

Parameters:

← ***sock*** socket descriptor

← ***pBuffer*** pointer to data to send

← ***size*** size of the data to send

← ***ipAddress*** destination IP

← ***port*** destination port

Return values:

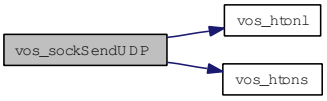
VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be sent

VOS_MEM_ERR resource error

Here is the call graph for this function:



5.23.2.20 EXT_DECL VOS_ERR_T vos_sockSetOptions (INT32 *sock*, const VOS_SOCK_OPT_T * *pOptions*)

Set socket options.

Note: Some targeted systems might not support every option.

Parameters:

← ***sock*** socket descriptor

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

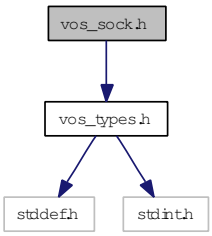
VOS_PARAM_ERR sock descriptor unknown

5.24 vos_sock.h File Reference

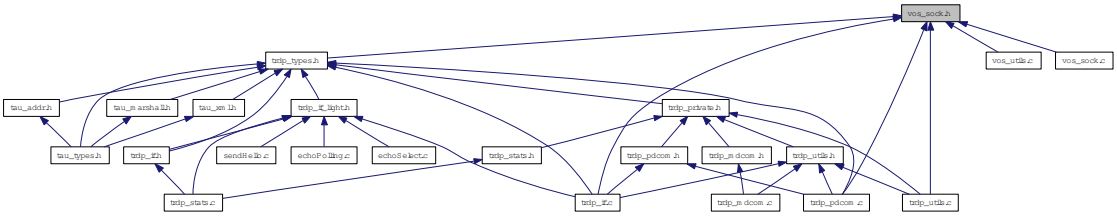
Typedefs for OS abstraction.

`#include "vos_types.h"`

Include dependency graph for vos_sock.h:



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



Data Structures

- struct VOS_SOCK_OPT_T

  Common socket options.

Defines

- #define VOS_MAX_SOCKET_CNT 80

  The maximum number of concurrent usable sockets.

- #define VOS_TTL_MULTICAST 64

  The maximum hops a multicast packet can go.

Functions

- EXT_DECL UINT16 vos_htons (UINT16 val)

  Byte swapping 2 Bytes.

- EXT_DECL UINT16 vos_ntohs (UINT16 val)

  Byte swapping 2 Bytes.

- EXT_DECL UINT32 vos_htonl (UINT32 val)

  Byte swapping 4 Bytes.

- EXT_DECL UINT32 vos_ntohl (UINT32 val)

  Byte swapping 4 Bytes.

- EXT_DECL BOOL vos_IsMulticast (UINT32 ipAddress)

  Check if the supplied address is a multicast group address.

- EXT_DECL VOS_ERR_T vos_sockInit (void)

  Initialize the socket library.

- EXT_DECL VOS_ERR_T vos_sockOpenUDP (INT32 *pSock, const VOS_SOCK_OPT_T *pOptions)

  Create an UDP socket.

- EXT_DECL VOS_ERR_T vos_sockOpenTCP (INT32 *pSock, const VOS_SOCK_OPT_T *pOptions)

  Create a TCP socket.

- EXT_DECL VOS_ERR_T vos_sockClose (INT32 sock)

  Close a socket.

- EXT_DECL VOS_ERR_T vos_sockSetOptions (INT32 sock, const VOS_SOCK_OPT_T *pOptions)

  Set socket options.

- EXT_DECL VOS_ERR_T vos_sockJoinMC (INT32 sock, UINT32 mcAddress, UINT32 ipAddress)

  Join a multicast group.

- EXT_DECL VOS_ERR_T vos_sockLeaveMC (INT32 sock, UINT32 mcAddress, UINT32 ipAddress)

  Leave a multicast group.

- EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 sock, const UINT8 *pBuffer, UINT32 size, UINT32 ipAddress, UINT16 port)

  Send UDP data.

- EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 sock, UINT8 *pBuffer, INT32 *pSize, UINT32 *pIPAddr)

  Receive UDP data.

- EXT_DECL VOS_ERR_T vos_sockBind (INT32 sock, UINT32 ipAddress, UINT16 port)

  Bind a socket to an address and port.

- EXT_DECL VOS_ERR_T vos_sockListen (INT32 sock, UINT32 backlog)

  Listen for incoming TCP connections.

- EXT_DECL VOS_ERR_T vos_sockAccept (INT32 sock, INT32 *pSock, UINT32 *pIPAddress, UINT16 *pPort)

  Accept an incoming TCP connection.

- EXT_DECL VOS_ERR_T vos_sockConnect (INT32 sock, UINT32 ipAddress, UINT16 port)

  Open a TCP connection.

- EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 sock, const UINT8 *pBuffer, UINT32 size)

  Send TCP data.

- EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 sock, UINT8 *pBuffer, INT32 *pSize)

  Receive TCP data.

5.24.1 Detailed Description

Typedefs for OS abstraction.

This is the declaration for the OS independend socket interface

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

vos_sock.h 2 2012-06-04 11:25:16Z 97025

5.24.2 Function Documentation

5.24.2.1 EXT_DECL UINT32 vos_htonl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

5.24.2.2 EXT_DECL UINT16 vos_htons (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

Byte swapping 2 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

5.24.2.3 EXT_DECL BOOL vos_IsMulticast (UINT32 *ipAddress*)

Check if the supplied address is a multicast group address.

Parameters:

← ***ipAddress*** IP address to check.

Return values:

TRUE address is multicast
FALSE address is not a multicast address

5.24.2.4 EXT_DECL UINT32 vos_ntohl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

5.24.2.5 EXT_DECL UINT16 vos_ntohs (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← ***val*** Initial value.

Return values:

swapped value

5.24.2.6 EXT_DECL VOS_ERR_T vos_sockAccept (INT32 *sock*, INT32 * *pSock*, UINT32 * *pIPAddress*, UINT16 * *pPort*)

Accept an incoming TCP connection.

Accept incoming connections on the provided socket. May block and will return a new socket descriptor when accepting a connection. The original socket *pSock, remains open.

Parameters:

← ***sock*** Socket descriptor

→ ***pSock*** Pointer to socket descriptor, on exit new socket

→ ***pIPAddress*** source IP to receive on, 0 for any

→ ***pPort*** port to receive on, 20548 for PD

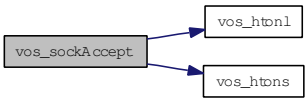
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR NULL parameter, parameter error

VOS_UNKNOWN_ERR sock descriptor unknown error

Here is the call graph for this function:



5.24.2.7 EXT_DECL VOS_ERR_T vos_sockBind (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Bind a socket to an address and port.

Parameters:

← ***sock*** socket descriptor

← ***ipAddress*** source IP to receive from, 0 for any

← ***port*** port to receive from

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

Parameters:

← ***sock*** socket descriptor

← ***ipAddress*** source IP to receive on, 0 for any

← ***port*** port to receive on, 20548 for PD

Return values:

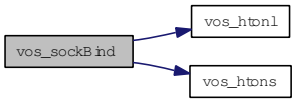
VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

Here is the call graph for this function:



5.24.2.8 EXT_DECL VOS_ERR_T vos_sockClose (INT32 *sock*)

Close a socket.

Release any resources aquired by this socket

Parameters:

← ***sock*** socket descriptor

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

Release any resources aquired by this socket

Parameters:

← ***sock*** socket descriptor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown

5.24.2.9 EXT_DECL VOS_ERR_T vos_sockConnect (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Open a TCP connection.

Parameters:

← ***sock*** socket descriptor

← ***ipAddress*** destination IP

← ***port*** destination port

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

Parameters:

← ***sock*** socket descriptor

← ***ipAddress*** destination IP

← ***port*** destination port

Return values:

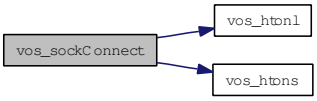
VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

Here is the call graph for this function:



5.24.2.10 EXT_DECL VOS_ERR_T vos_sockInit (void)

Initialize the socket library.

Must be called once before any other call

Return values:

VOS_NO_ERR no error

VOS_SOCK_ERR sockets not supported

5.24.2.11 EXT_DECL VOS_ERR_T vos_sockJoinMC (INT32 *sock*, UINT32 *mcAddress*, UINT32 *ipAddress*)

Join a multicast group.

Note: Some target systems might not support this option.

Parameters:

← ***sock*** socket descriptor

← ***mcAddress*** multicast group to join

← ***ipAddress*** depicts interface on which to join, default 0 for any

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_SOCK_ERR option not supported

Note: Some targeted systems might not support this option.

Parameters:

← ***sock*** socket descriptor

← ***mcAddress*** multicast group to join

← ***ipAddress*** depicts interface on which to join, default 0 for any

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_SOCK_ERR option not supported

Here is the call graph for this function:

5.24.2.12 EXT_DECL VOS_ERR_T vos_sockLeaveMC (INT32 *sock*, UINT32 *mcAddress*, UINT32 *ipAddress*)

Leave a multicast group.

Note: Some target systems might not support this option.

Parameters:

← ***sock*** socket descriptor

← ***mcAddress*** multicast group to join

← ***ipAddress*** depicts interface on which to leave, default 0 for any

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_SOCK_ERR option not supported

Note: Some targeted systems might not support this option.

Parameters:

← ***sock*** socket descriptor

← ***mcAddress*** multicast group to join

← ***ipAddress*** depicts interface on which to leave, default 0 for any

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_SOCK_ERR option not supported

Here is the call graph for this function:

5.24.2.13 EXT_DECL VOS_ERR_T vos_sockListen (INT32 *sock*, UINT32 *backlog*)

Listen for incoming TCP connections.

Parameters:

← ***sock*** socket descriptor

← ***backlog*** maximum connection attempts if system is busy

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

Listen for incoming TCP connections.

Parameters:

← ***sock*** socket descriptor

← ***backlog*** maximum connection attempts if system is busy

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

5.24.2.14 EXT_DECL VOS_ERR_T vos_sockOpenTCP (INT32 * *pSock*, const VOS_SOCK_OPT_T * *pOptions*)

Create a TCP socket.

Return a socket descriptor for further calls. The socket options are optional and can be applied later.

Parameters:

→ ***pSock*** pointer to socket descriptor returned

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR pSock == NULL

VOS_SOCK_ERR socket not available or option not supported

Return a socket descriptor for further calls. The socket options are optional and can be applied later.

Parameters:

→ ***pSock*** pointer to socket descriptor returned

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pSock == NULL

VOS_SOCK_ERR socket not available or option not supported

Here is the call graph for this function:

5.24.2.15 EXT_DECL VOS_ERR_T vos_sockOpenUDP (INT32 * *pSock*, const VOS_SOCK_OPT_T * *pOptions*)

Create an UDP socket.

Return a socket descriptor for further calls. The socket options are optional and can be applied later. Note: Some target systems might not support every option.

Parameters:

→ ***pSock*** pointer to socket descriptor returned

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pSock == NULL

VOS_SOCK_ERR socket not available or option not supported

Return a socket descriptor for further calls. The socket options are optional and can be applied later. Note: Some targeted systems might not support every option.

Parameters:

→ ***pSock*** pointer to socket descriptor returned

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pSock == NULL

VOS_SOCK_ERR socket not available or option not supported

Here is the call graph for this function:

5.24.2.16 EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 *sock*, UINT8 * *pBuffer*, INT32 * *pSize*)

Receive TCP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error occured. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned.

Parameters:

← ***sock*** socket descriptor

→ ***pBuffer*** pointer to applications data buffer

↔ ***pSize*** pointer to the received data size

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_IO_ERR data could not be read

VOS_MEM_ERR resource error

VOS_NODATA_ERR no data in non-blocking

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, ∗pSize will reflect the number of copied bytes and the call should be repeated until ∗pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error occured. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned.

Parameters:

← ***sock*** socket descriptor

→ ***pBuffer*** pointer to applications data buffer

↔ ***pSize*** pointer to the received data size

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be read

VOS_NODATA_ERR no data in non-blocking

5.24.2.17 EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 *sock*, UINT8 ∗ *pBuffer*, INT32 ∗ *pSize*, UINT32 ∗ *pIPAddr*)

Receive UDP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, ∗pSize will reflect the number of copied bytes and the call should be repeated until ∗pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error occured. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned.

Parameters:

← ***sock*** socket descriptor

→ ***pBuffer*** pointer to applications data buffer

↔ ***pSize*** pointer to the received data size

→ ***pIPAddr*** source IP

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_IO_ERR data could not be read

VOS_MEM_ERR resource error

VOS_NODATA_ERR no data in non-blocking

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, ∗pSize will reflect the number of copied bytes and the call should be repeated until ∗pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error occured. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned.

Parameters:

← ***sock*** socket descriptor

→ ***pBuffer*** pointer to applications data buffer

↔ ***pSize*** pointer to the received data size

→ ***pIPAddr*** source IP

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be read

VOS_NODATA_ERR no data in non-blocking

Here is the call graph for this function:

5.24.2.18 EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 *sock*, const UINT8 ∗ *pBuffer*, UINT32 *size*)

Send TCP data.

Send data to the given socket.

Parameters:

← ***sock*** socket descriptor

← ***pBuffer*** pointer to data to send

← ***size*** size of the data to send

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_IO_ERR data could not be sent

VOS_MEM_ERR resource error

Send data to the supplied address and port.

Parameters:

← ***sock*** socket descriptor

← ***pBuffer*** pointer to data to send

← ***size*** size of the data to send

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be sent

5.24.2.19 EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 *size*, UINT32 *ipAddress*, UINT16 *port*)

Send UDP data.

Send data to the given address and port.

Parameters:

← ***sock*** socket descriptor

← ***pBuffer*** pointer to data to send

← ***size*** size of the data to send

← ***ipAddress*** destination IP

← ***port*** destination port

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_IO_ERR data could not be sent

VOS_MEM_ERR resource error

Send data to the supplied address and port.

Parameters:

← ***sock*** socket descriptor

← ***pBuffer*** pointer to data to send

← ***size*** size of the data to send

← ***ipAddress*** destination IP

← ***port*** destination port

Return values:

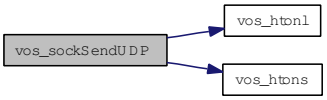
VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be sent

VOS_MEM_ERR resource error

Here is the call graph for this function:



5.24.2.20 EXT_DECL VOS_ERR_T vos_sockSetOptions (INT32 *sock*, const VOS_SOCK_OPT_T * *pOptions*)

Set socket options.

Note: Some target systems might not support each option.

Parameters:

← ***sock*** socket descriptor

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_SOCK_ERR socket not available or option not supported

Note: Some targeted systems might not support every option.

Parameters:

← ***sock*** socket descriptor

← ***pOptions*** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

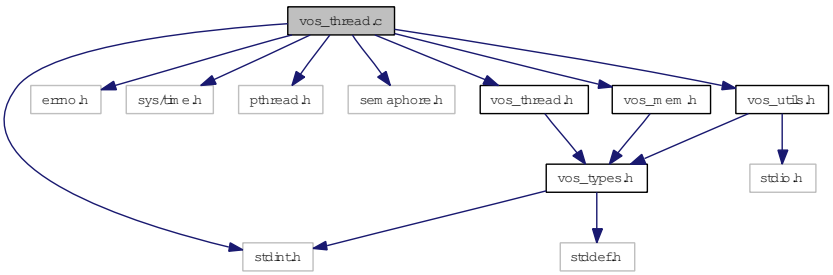
VOS_PARAM_ERR sock descriptor unknown

5.25 vos_thread.c File Reference

Multitasking functions.

```
#include <stdint.h>
#include <errno.h>
#include <sys/time.h>
#include <pthread.h>
#include <semaphore.h>
#include "vos_thread.h"
#include "vos_mem.h"
#include "vos_utils.h"
```

Include dependency graph for vos_thread.c:



Functions

- void cyclicThread (UINT32 interval, VOS_THREAD_FUNC_T pFunction, void *pArguments)

  Cyclic thread functions.

- EXT_DECL VOS_ERR_T vos_threadInit (void)

  Initialize the thread library.

- EXT_DECL VOS_ERR_T vos_threadCreate (VOS_THREAD_T *pThread, const CHAR8 *pName, VOS_THREAD_POLICY_T policy, VOS_THREAD_PRIORITY_T priority, UINT32 interval, UINT32 stackSize, VOS_THREAD_FUNC_T pFunction, void *pArguments)

  Create a thread.

- EXT_DECL VOS_ERR_T vos_threadTerminate (VOS_THREAD_T thread)

  Terminate a thread.

- EXT_DECL VOS_ERR_T vos_threadIsActive (VOS_THREAD_T thread)

  Is the thread still active? This call will return VOS_NO_ERR if the thread is still active, VOS_PARAM_ERR in case it ran out.

- EXT_DECL VOS_ERR_T vos_threadDelay (UINT32 delay)

  Delay the execution of the current thread by the given delay in us.

- EXT_DECL VOS_ERR_T vos_getTime (VOS_TIME_T *pTime)

  Return the current time in sec and us.

- EXT_DECL const CHAR8 * vos_getTimeStamp (void)

  Get a time-stamp string.

- EXT_DECL VOS_ERR_T vos_clearTime (VOS_TIME_T *pTime)

  Clear the time stamp.

- EXT_DECL VOS_ERR_T vos_addTime (VOS_TIME_T *pTime, const VOS_TIME_T *pAdd)

  Add the second to the first time stamp, return sum in first.

- EXT_DECL VOS_ERR_T vos_subTime (VOS_TIME_T *pTime, const VOS_TIME_T *pSub)

  Subtract the second from the first time stamp, return diff in first.

- EXT_DECL INT32 vos_cmpTime (const VOS_TIME_T *pTime, const VOS_TIME_T *pCmp)

  Compare the second from the first time stamp, return diff in first.

- EXT_DECL VOS_ERR_T vos_getUuid (VOS_UUID_T pUuID)

  Get a universal unique identifier according to RFC 4122 time based version.

- EXT_DECL VOS_ERR_T vos_mutexCreate (VOS_MUTEX_T *pMutex)

  Create a recursive mutex.

- EXT_DECL VOS_ERR_T vos_mutexDelete (VOS_MUTEX_T mutex)

  Delete a mutex.

- EXT_DECL VOS_ERR_T vos_mutexLock (VOS_MUTEX_T mutex)

  Take a mutex.

- EXT_DECL VOS_ERR_T vos_mutexTryLock (VOS_MUTEX_T mutex)

  Try to take a mutex.

- EXT_DECL VOS_ERR_T vos_mutexUnlock (VOS_MUTEX_T mutex)

  Release a mutex.

- EXT_DECL VOS_ERR_T vos_semaCreate (VOS_SEMA_T *pSema, VOS_SEMA_STATE_T initialState)

  Create a semaphore.

- EXT_DECL VOS_ERR_T vos_semaDelete (VOS_SEMA_T sema)

  Delete a semaphore.

- EXT_DECL VOS_ERR_T vos_semaTake (VOS_SEMA_T sema, UINT32 timeout)

  Take a semaphore.

- EXT_DECL VOS_ERR_T vos_semaGive (VOS_SEMA_T sema)

  Give a semaphore.

5.25.1 Detailed Description

Multitasking functions.

OS abstraction of thread-handling functions

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

vos_thread.c 2 2012-06-04 11:25:16Z 97025

5.25.2 Function Documentation

5.25.2.1 void cyclicThread (UINT32 *interval*, VOS_THREAD_FUNC_T *pFunction*, void * *pArguments*)

Cyclic thread functions.

Wrapper for cyclic threads. The thread function will be called cyclically with interval.

Parameters:

← ***interval*** Interval for cyclic threads in us (optional)

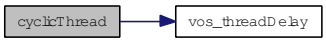
← ***pFunction*** Pointer to the thread function

← ***pArguments*** Pointer to the thread function parameters

Return values:

void

Here is the call graph for this function:



5.25.2.2 EXT_DECL VOS_ERR_T vos_addTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pAdd*)

Add the second to the first time stamp, return sum in first.

Parameters:

↔ ***pTime*** Pointer to time value

← ***pAdd*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.25.2.3 EXT_DECL VOS_ERR_T vos_clearTime (VOS_TIME_T * *pTime*)

Clear the time stamp.

Parameters:

→ ***pTime*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.25.2.4 EXT_DECL INT32 vos_cmpTime (const VOS_TIME_T * *pTime*, const VOS_TIME_T * *pCmp*)

Compare the second from the first time stamp, return diff in first.

Parameters:

↔ ***pTime*** Pointer to time value

← ***pCmp*** Pointer to time value to compare

Return values:

0 pTime == pCmp

-1 pTime < pCmp

1 pTime > pCmp

5.25.2.5 EXT_DECL VOS_ERR_T vos_getTime (VOS_TIME_T * *pTime*)

Return the current time in sec and us.

Parameters:

→ ***pTime*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

5.25.2.6 EXT_DECL const CHAR8* vos_getTimeStamp (void)

Get a time-stamp string.

Get a time-stamp string for debugging in the form "yyyymmdd-hh:mm:ss.ms" Depending on the used OS / hardware the time might not be a real-time stamp but relative from start of system.

Return values:

timestamp "yyyymmdd-hh:mm:ss.ms"

5.25.2.7 EXT_DECL VOS_ERR_T vos_getUuid (VOS_UUID_T *pUuID*)

Get a universal unique identifier according to RFC 4122 time based version.

Parameters:

→ ***pUuID*** Pointer to a universal unique identifier

Return values:

VOS_NO_ERR no error
VOS_INIT_ERR module not initialised

Here is the call graph for this function:

5.25.2.8 EXT_DECL VOS_ERR_T vos_mutexCreate (VOS_MUTEX_T * *pMutex*)

Create a recursive mutex.

Create a mutex.

Return a mutex handle. The mutex will be available at creation.

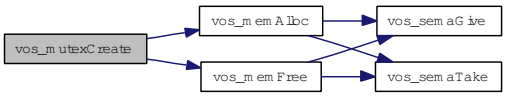
Parameters:

→ ***pMutex*** Pointer to mutex handle

Return values:

VOS_NO_ERR no error
VOS_INIT_ERR module not initialised
VOS_PARAM_ERR pMutex == NULL
VOS_MUTEX_ERR no mutex available

Here is the call graph for this function:



5.25.2.9 EXT_DECL VOS_ERR_T vos_mutexDelete (VOS_MUTEX_T *mutex*)

Delete a mutex.

Release the resources taken by the mutex.

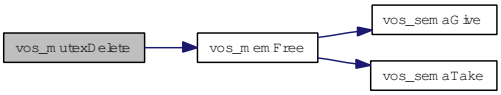
Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR pMutex == NULL or wrong type
VOS_MUTEX_ERR no such mutex

Here is the call graph for this function:



5.25.2.10 EXT_DECL VOS_ERR_T vos_mutexLock (VOS_MUTEX_T *mutex*)

Take a mutex.

Wait for the mutex to become available (lock).

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR pMutex == NULL or wrong type
VOS_MUTEX_ERR no such mutex

5.25.2.11 EXT_DECL VOS_ERR_T vos_mutexTryLock (VOS_MUTEX_T *mutex*)

Try to take a mutex.

If mutex is can't be taken VOS_MUTEX_ERR is returned.

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR pMutex == NULL or wrong type
VOS_MUTEX_ERR mutex not locked

5.25.2.12 EXT_DECL VOS_ERR_T vos_mutexUnlock (VOS_MUTEX_T *mutex*)

Release a mutex.

Unlock the mutex.

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMutex == NULL or wrong type

VOS_MUTEX_ERR no such mutex

5.25.2.13 EXT_DECL VOS_ERR_T vos_semaCreate (VOS_SEMA_T * *pSema*, VOS_SEMA_STATE_T *initialState*)

Create a semaphore.

Return a semaphore handle. Depending on the initial state the semaphore will be available on creation or not.

Parameters:

→ ***pSema*** Pointer to semaphore handle

← ***initialState*** The initial state of the sempahore

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

VOS_SEMA_ERR no semaphore available

5.25.2.14 EXT_DECL VOS_ERR_T vos_semaDelete (VOS_SEMA_T *sema*)

Delete a semaphore.

This will eventually release any processes waiting for the semaphore.

Parameters:

← ***sema*** semaphore handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

5.25.2.15 EXT_DECL VOS_ERR_T vos_semaGive (VOS_SEMA_T *sema*)

Give a semaphore.

Release (increase) a semaphore.

Parameters:

← ***sema*** semaphore handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_SEM_ERR could not release semaphore

5.25.2.16 EXT_DECL VOS_ERR_T vos_semaTake (VOS_SEMA_T *sema*, UINT32 *timeout*)

Take a semaphore.

Try to get (decrease) a semaphore.

Parameters:

← ***sema*** semaphore handle

← ***timeout*** Max. time in us to wait, 0 means forever

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_SEMA_ERR could not get semaphore in time

5.25.2.17 EXT_DECL VOS_ERR_T vos_subTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pSub*)

Subtract the second from the first time stamp, return diff in first.

Parameters:

↔ ***pTime*** Pointer to time value

← ***pSub*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.25.2.18 EXT_DECL VOS_ERR_T vos_threadCreate (VOS_THREAD_T * *pThread*, const CHAR8 * *pName*, VOS_THREAD_POLICY_T *policy*, VOS_THREAD_PRIORITY_T *priority*, UINT32 *interval*, UINT32 *stackSize*, VOS_THREAD_FUNC_T *pFunction*, void * *pArguments*)

Create a thread.

Create a thread and return a thread handle for further requests. Not each parameter may be supported by all target systems!

Parameters:

→ ***pThread*** Pointer to returned thread handle
← ***pName*** Pointer to name of the thread (optional)
← ***policy*** Scheduling policy (FIFO, Round Robin or other)
← ***priority*** Scheduling priority (1...255 (highest), default 0)
← ***interval*** Interval for cyclic threads in us (optional)
← ***stackSize*** Minimum stacksize, default 0: 16kB
← ***pFunction*** Pointer to the thread function
← ***pArguments*** Pointer to the thread function parameters

Return values:

VOS_NO_ERR no error
VOS_INIT_ERR module not initialised
VOS_NOINIT_ERR invalid handle
VOS_PARAM_ERR parameter out of range/invalid
VOS_THREAD_ERR thread creation error
VOS_INIT_ERR no threads available

5.25.2.19 EXT_DECL VOS_ERR_T vos_threadDelay (UINT32 *delay*)

Delay the execution of the current thread by the given delay in us.

Parameters:

← ***delay*** Delay in us

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR parameter out of range/invalid

5.25.2.20 EXT_DECL VOS_ERR_T vos_threadInit (void)

Initialize the thread library.

Must be called once before any other call

Return values:

VOS_NO_ERR no error
VOS_INIT_ERR threading not supported

5.25.2.21 EXT_DECL VOS_ERR_T vos_threadIsActive (VOS_THREAD_T *thread*)

Is the thread still active? This call will return VOS_NO_ERR if the thread is still active, VOS_PARAM_-ERR in case it ran out.

Parameters:

← ***thread*** Thread handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

5.25.2.22 EXT_DECL VOS_ERR_T vos_threadTerminate (VOS_THREAD_T *thread*)

Terminate a thread.

This call will terminate the thread with the given threadId and release all resources. Depending on the underlying architectures, it may just block until the thread ran out.

Parameters:

← ***thread*** Thread handle (or NULL if current thread)

Return values:

VOS_NO_ERR no error

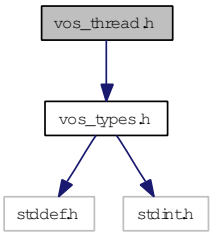
VOS_THREAD_ERR cancel failed

5.26 vos_thread.h File Reference

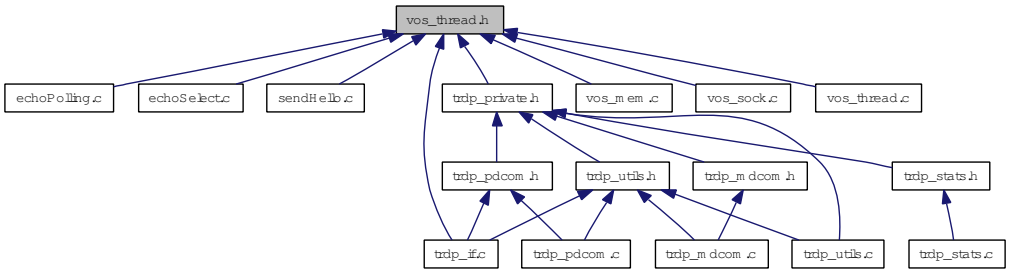
Threading functions for OS abstraction.

```
#include "vos_types.h"
```

Include dependency graph for vos_thread.h:



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



Typedefs

- typedef UINT8 VOS_THREAD_PRIORITY_T

  Thread priority range from 1 (highest) to 255 (lowest), 0 default of the target system.

- typedef void(__cdecl * VOS_THREAD_FUNC_T)(void *pArg)

  Thread function definition.

- typedef struct VOS_MUTEX_T * VOS_MUTEX_T

  Hidden mutex handle definition.

- typedef struct VOS_SEMA_T * VOS_SEMA_T

  Hidden semaphore handle definition.

- typedef void * VOS_THREAD_T

  Hidden thread handle definition.

Enumerations

- enum VOS_THREAD_POLICY_T

Thread policy matching pthread/Posix defines.

- enum VOS_SEMA_STATE_T

  State of the semaphore.

Functions

- EXT_DECL VOS_ERR_T vos_threadInit (void)

  Initialize the thread library.

- EXT_DECL VOS_ERR_T vos_threadCreate (VOS_THREAD_T *pThread, const CHAR8 *pName, VOS_THREAD_POLICY_T policy, VOS_THREAD_PRIORITY_T priority, UINT32 interval, UINT32 stackSize, VOS_THREAD_FUNC_T pFunction, void *pArguments)

  Create a thread.

- EXT_DECL VOS_ERR_T vos_threadTerminate (VOS_THREAD_T thread)

  Terminate a thread.

- EXT_DECL VOS_ERR_T vos_threadIsActive (VOS_THREAD_T thread)

  Is the thread still active? This call will return VOS_NO_ERR if the thread is still active, VOS_PARAM_ERR in case it ran out.

- EXT_DECL VOS_ERR_T vos_threadDelay (UINT32 delay)

  Delay the execution of the current thread by the given delay in us.

- EXT_DECL VOS_ERR_T vos_getTime (VOS_TIME_T *pTime)

  Return the current time in sec and us.

- EXT_DECL const CHAR8 * vos_getTimeStamp (void)

  Get a time-stamp string.

- EXT_DECL VOS_ERR_T vos_clearTime (VOS_TIME_T *pTime)

  Clear the time stamp.

- EXT_DECL VOS_ERR_T vos_addTime (VOS_TIME_T *pTime, const VOS_TIME_T *pAdd)

  Add the second to the first time stamp, return sum in first.

- EXT_DECL VOS_ERR_T vos_subTime (VOS_TIME_T *pTime, const VOS_TIME_T *pSub)

  Subtract the second from the first time stamp, return diff in first.

- EXT_DECL INT32 vos_cmpTime (const VOS_TIME_T *pTime, const VOS_TIME_T *pCmp)

  Compare the second from the first time stamp, return diff in first.

- EXT_DECL VOS_ERR_T vos_getUuid (VOS_UUID_T pUuID)

  Get a universal unique identifier according to RFC 4122 time based version.

- EXT_DECL VOS_ERR_T vos_mutexCreate (VOS_MUTEX_T *pMutex)

  Create a mutex.

- EXT_DECL VOS_ERR_T vos_mutexDelete (VOS_MUTEX_T mutex)

  Delete a mutex.

- EXT_DECL VOS_ERR_T vos_mutexLock (VOS_MUTEX_T mutex)

  Take a mutex.

- EXT_DECL VOS_ERR_T vos_mutexTryLock (VOS_MUTEX_T mutex)

  Try to take a mutex.

- EXT_DECL VOS_ERR_T vos_mutexUnlock (VOS_MUTEX_T mutex)

  Release a mutex.

- EXT_DECL VOS_ERR_T vos_semaCreate (VOS_SEMA_T *pSema, VOS_SEMA_STATE_T initialState)

  Create a semaphore.

- EXT_DECL VOS_ERR_T vos_semaDelete (VOS_SEMA_T sema)

  Delete a semaphore.

- EXT_DECL VOS_ERR_T vos_semaTake (VOS_SEMA_T sema, UINT32 timeout)

  Take a semaphore.

- EXT_DECL VOS_ERR_T vos_semaGive (VOS_SEMA_T sema)

  Give a semaphore.

5.26.1 Detailed Description

Threading functions for OS abstraction.

Thread-, semaphore- and time-handling functions

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

vos_thread.h 2 2012-06-04 11:25:16Z 97025

5.26.2 Function Documentation

5.26.2.1 EXT_DECL VOS_ERR_T vos_addTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pAdd*)

Add the second to the first time stamp, return sum in first.

Parameters:

↔ ***pTime*** Pointer to time value

← ***pAdd*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.26.2.2 EXT_DECL VOS_ERR_T vos_clearTime (VOS_TIME_T * *pTime*)

Clear the time stamp.

Parameters:

→ ***pTime*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.26.2.3 EXT_DECL INT32 vos_cmpTime (const VOS_TIME_T * *pTime*, const VOS_TIME_T * *pCmp*)

Compare the second from the first time stamp, return diff in first.

Parameters:

↔ ***pTime*** Pointer to time value

← ***pCmp*** Pointer to time value to compare

Return values:

0 pTime == pCmp

-1 pTime < pCmp

1 pTime > pCmp

5.26.2.4 EXT_DECL VOS_ERR_T vos_getTime (VOS_TIME_T * *pTime*)

Return the current time in sec and us.

Parameters:

→ ***pTime*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

Parameters:

→ ***pTime*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

5.26.2.5 EXT_DECL const CHAR8* vos_getTimeStamp (void)

Get a time-stamp string.

Get a time-stamp string for debugging in the form "yyyymmdd-hh:mm:ss.ms" Depending on the used OS / hardware the time might not be a real-time stamp but relative from start of system.

Return values:

timestamp "yyyymmdd-hh:mm:ss.ms"

5.26.2.6 EXT_DECL VOS_ERR_T vos_getUuid (VOS_UUID_T *pUuID*)

Get a universal unique identifier according to RFC 4122 time based version.

Parameters:

→ ***pUuID*** Pointer to a universal unique identifier

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

Here is the call graph for this function:

5.26.2.7 EXT_DECL VOS_ERR_T vos_mutexCreate (VOS_MUTEX_T * *pMutex*)

Create a mutex.

Return a mutex handle. The mutex will be available at creation.

Parameters:

→ ***pMutex*** Pointer to mutex handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR pMutex == NULL

VOS_MUTEX_ERR no mutex available

Create a mutex.

Return a mutex handle. The mutex will be available at creation.

Parameters:

→ ***pMutex*** Pointer to mutex handle

Return values:

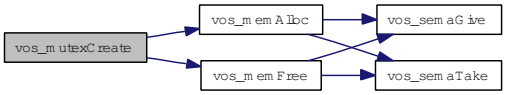
VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR pMutex == NULL

VOS_MUTEX_ERR no mutex available

Here is the call graph for this function:



5.26.2.8 EXT_DECL VOS_ERR_T vos_mutexDelete (VOS_MUTEX_T *mutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_MUTEX_ERR no such mutex

Release the resources taken by the mutex.

Parameters:

← ***mutex*** mutex handle

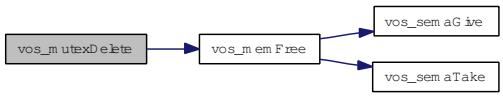
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMutex == NULL or wrong type

VOS_MUTEX_ERR no such mutex

Here is the call graph for this function:



5.26.2.9 EXT_DECL VOS_ERR_T vos_mutexLock (VOS_MUTEX_T *mutex*)

Take a mutex.

Wait for the mutex to become available (lock).

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

Wait for the mutex to become available (lock).

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMutex == NULL or wrong type

VOS_MUTEX_ERR no such mutex

5.26.2.10 EXT_DECL VOS_ERR_T vos_mutexTryLock (VOS_MUTEX_T *mutex*)

Try to take a mutex.

If mutex is can't be taken VOS_MUTEX_ERR is returned.

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error
VOS_INIT_ERR module not initialised
VOS_NOINIT_ERR invalid handle
VOS_MUTEX_ERR no mutex available

If mutex is can't be taken VOS_MUTEX_ERR is returned.

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR pMutex == NULL or wrong type
VOS_MUTEX_ERR mutex not locked

5.26.2.11 EXT_DECL VOS_ERR_T vos_mutexUnlock (VOS_MUTEX_T *mutex*)

Release a mutex.

Unlock the mutex.

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error
VOS_INIT_ERR module not initialised
VOS_NOINIT_ERR invalid handle

Unlock the mutex.

Parameters:

← ***mutex*** mutex handle

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR pMutex == NULL or wrong type
VOS_MUTEX_ERR no such mutex

5.26.2.12 EXT_DECL VOS_ERR_T vos_semaCreate (VOS_SEMA_T * *pSema*, VOS_SEMA_STATE_T *initialState*)

Create a semaphore.

Return a semaphore handle. Depending on the initial state the semaphore will be available on creation or not.

Parameters:

→ ***pSema*** Pointer to semaphore handle

← ***initialState*** The initial state of the sempahore

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

VOS_SEMA_ERR no semaphore available

5.26.2.13 EXT_DECL VOS_ERR_T vos_semaDelete (VOS_SEMA_T *sema*)

Delete a semaphore.

This will eventually release any processes waiting for the semaphore.

Parameters:

← ***sema*** semaphore handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

5.26.2.14 EXT_DECL VOS_ERR_T vos_semaGive (VOS_SEMA_T *sema*)

Give a semaphore.

Release (increase) a semaphore.

Parameters:

← ***sema*** semaphore handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_SEM_ERR could not release semaphore

5.26.2.15 EXT_DECL VOS_ERR_T vos_semaTake (VOS_SEMA_T *sema*, UINT32 *timeout*)

Take a semaphore.

Try to get (decrease) a semaphore.

Parameters:

← ***sema*** semaphore handle

← ***timeout*** Max. time in us to wait, 0 means forever

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_SEMA_ERR could not get semaphore in time

5.26.2.16 EXT_DECL VOS_ERR_T vos_subTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pSub*)

Subtract the second from the first time stamp, return diff in first.

Parameters:

↔ ***pTime*** Pointer to time value

← ***pSub*** Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.26.2.17 EXT_DECL VOS_ERR_T vos_threadCreate (VOS_THREAD_T * *pThread*, const CHAR8 * *pName*, VOS_THREAD_POLICY_T *policy*, VOS_THREAD_PRIORITY_T *priority*, UINT32 *interval*, UINT32 *stackSize*, VOS_THREAD_FUNC_T *pFunction*, void * *pArguments*)

Create a thread.

Create a thread and return a thread handle for further requests. Not each parameter may be supported by all target systems!

Parameters:

→ ***pThread*** Pointer to returned thread handle

← ***pName*** Pointer to name of the thread (optional)

← ***policy*** Scheduling policy (FIFO, Round Robin or other)

← ***priority*** Scheduling priority (1...255 (highest), default 0)

← ***interval*** Interval for cyclic threads in us (optional)

← ***stackSize*** Minimum stacksize, default 0: 16kB

← ***pFunction*** Pointer to the thread function

← ***pArguments*** Pointer to the thread function parameters

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_INIT_ERR no threads available

Create a thread and return a thread handle for further requests. Not each parameter may be supported by all target systems!

Parameters:

→ ***pThread*** Pointer to returned thread handle

← ***pName*** Pointer to name of the thread (optional)

← ***policy*** Scheduling policy (FIFO, Round Robin or other)

← ***priority*** Scheduling priority (1...255 (highest), default 0)

← ***interval*** Interval for cyclic threads in us (optional)

← ***stackSize*** Minimum stacksize, default 0: 16kB

← ***pFunction*** Pointer to the thread function

← ***pArguments*** Pointer to the thread function parameters

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_THREAD_ERR thread creation error

VOS_INIT_ERR no threads available

5.26.2.18 EXT_DECL VOS_ERR_T vos_threadDelay (UINT32 *delay*)

Delay the execution of the current thread by the given delay in us.

Parameters:

← ***delay*** Delay in us

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

Parameters:

← ***delay*** Delay in us

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

5.26.2.19 EXT_DECL VOS_ERR_T vos_threadInit (void)

Initialize the thread library.

Must be called once before any other call

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR threading not supported

5.26.2.20 EXT_DECL VOS_ERR_T vos_threadIsActive (VOS_THREAD_T *thread*)

Is the thread still active? This call will return VOS_NO_ERR if the thread is still active, VOS_PARAM_-ERR in case it ran out.

Parameters:

← ***thread*** Thread handle

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

Parameters:

← ***thread*** Thread handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

5.26.2.21 EXT_DECL VOS_ERR_T vos_threadTerminate (VOS_THREAD_T *thread*)

Terminate a thread.

This call will terminate the thread with the given threadId and release all resources. Depending on the underlying architectures, it may just block until the thread ran out.

Parameters:

← ***thread*** Thread handle (or NULL if current thread)

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

This call will terminate the thread with the given threadId and release all resources. Depending on the underlying architectures, it may just block until the thread ran out.

Parameters:

← ***thread*** Thread handle (or NULL if current thread)

Return values:

VOS_NO_ERR no error

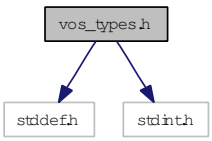
VOS_THREAD_ERR cancel failed

5.27 vos_types.h File Reference

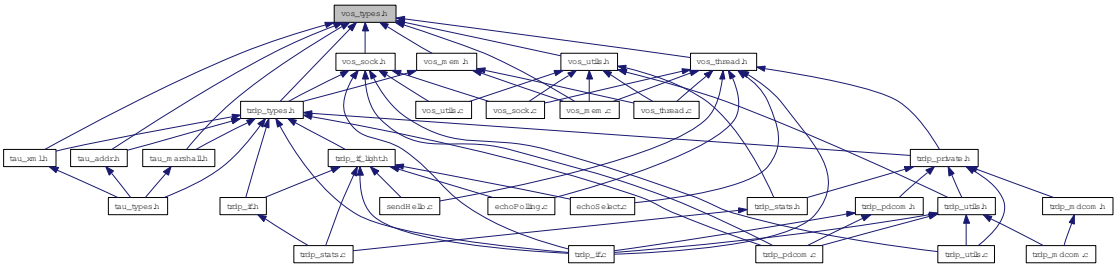
Typedefs for OS abstraction.

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for vos_types.h:



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



Data Structures

- struct VOS_TIME_T

  Timer value compatible with timeval / select.

Typedefs

- typedef UINT8 VOS_UUID_T [16]

  universal unique identifier according to RFC 4122, time based version

- typedef void(∗ VOS_PRINT_DBG_T)(void ∗pRefCon, VOS_LOG_T category, const CHAR8 ∗pTime, const CHAR8 ∗pFile, UINT16 LineNumber, const CHAR8 ∗pMsgStr)

  Function definition for error/debug output.

Enumerations

- enum VOS_ERR_T {
  VOS_NO_ERR = 0,
  VOS_PARAM_ERR = -1,
  VOS_INIT_ERR = -2,

VOS_NOINIT_ERR = -3,
VOS_TIMEOUT_ERR = -4,
VOS_NODATA_ERR = -5,
VOS_SOCK_ERR = -6,
VOS_IO_ERR = -7,
VOS_MEM_ERR = -8,
VOS_SEMA_ERR = -9,
VOS_QUEUE_ERR = -10,
VOS_QUEUE_FULL_ERR = -11,
VOS_MUTEX_ERR = -12,
VOS_THREAD_ERR = -13,
VOS_UNKNOWN_ERR = -99 }
Return codes for all VOS API functions.

- enum VOS_LOG_T {
VOS_LOG_ERROR = 0,
VOS_LOG_WARNING = 1,
VOS_LOG_INFO = 2,
VOS_LOG_DBG = 3 }
Categories for logging.

Functions

- EXT_DECL VOS_ERR_T vos_init (void *pRefCon, VOS_PRINT_DBG_T pDebugOutput)
Initialize the vos library.

5.27.1 Detailed Description

Typedefs for OS abstraction.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

vos_types.h 2 2012-06-04 11:25:16Z 97025

5.27.2 Typedef Documentation

5.27.2.1 typedef void(∗ VOS_PRINT_DBG_T)(void ∗pRefCon, VOS_LOG_T category, const CHAR8 ∗pTime, const CHAR8 ∗pFile, UINT16 LineNumber, const CHAR8 ∗pMsgStr)

Function definition for error/debug output.

The function will be called for logging and error message output. The user can decide, what kind of info will be logged by filtering the category.

Parameters:

← ***∗pRefCon*** pointer to user context

← ***category*** Log category (Error, Warning, Info etc.)

← ***pTime*** pointer to NULL-terminated string of time stamp

← ***pFile*** pointer to NULL-terminated string of source module

← ***LineNumber*** Line number

← ***pMsgStr*** pointer to NULL-terminated string

Return values:

none

5.27.3 Enumeration Type Documentation

5.27.3.1 enum VOS_ERR_T

Return codes for all VOS API functions.

Enumerator:

VOS_NO_ERR No error.

VOS_PARAM_ERR Necessary parameter missing or out of range.

VOS_INIT_ERR Call without valid initialization.

VOS_NOINIT_ERR The supplied handle/reference is not valid.

VOS_TIMEOUT_ERR Timout.

VOS_NODATA_ERR Non blocking mode: no data received.

VOS_SOCK_ERR Socket option not supported.

VOS_IO_ERR Socket IO error, data can't be received/sent.

VOS_MEM_ERR No more memory available.

VOS_SEMA_ERR Semaphore not available.

VOS_QUEUE_ERR Queue empty.

VOS_QUEUE_FULL_ERR Queue full.

VOS_MUTEX_ERR Mutex not available.

VOS_THREAD_ERR Thread creation error.

VOS_UNKNOWN_ERR Unknown error.

5.27.3.2 enum VOS_LOG_T

Categories for logging.

Enumerator:

VOS_LOG_ERROR This is a critical error.

VOS_LOG_WARNING This is a warning.

VOS_LOG_INFO This is an info.

VOS_LOG_DBG This is a debug info.

5.27.4 Function Documentation

5.27.4.1 EXT_DECL VOS_ERR_T vos_init (void * *pRefCon*, VOS_PRINT_DBG_T *pDebugOutput*)

Initialize the vos library.

This is used to set the output function for all VOS error and debug output.

Parameters:

← ∗***pRefCon*** user context

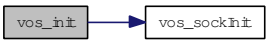
← ∗***pDebugOutput*** pointer to debug output function

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR unsupported

Here is the call graph for this function:

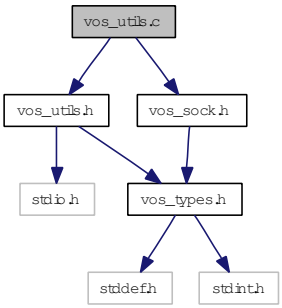


5.28 vos_utils.c File Reference

Common functions for VOS.

```
#include "vos_utils.h"
#include "vos_sock.h"
```

Include dependency graph for vos_utils.c:



Functions

- VOS_ERR_T vos_init (void *pRefCon, VOS_PRINT_DBG_T pDebugOutput)

  Initialize the vos library.

- UINT32 vos_crc32 (UINT32 crc, const UINT8 *pData, UINT32 dataLen)

  Compute crc32 according to IEEE802.3.

5.28.1 Detailed Description

Common functions for VOS.

Common functions of the abstraction layer. Mainly debugging support.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

vos_utils.c 2 2012-06-04 11:25:16Z 97025

5.28.2 Function Documentation

5.28.2.1 UINT32 vos_crc32 (UINT32 *crc*, const UINT8 * *pData*, UINT32 *dataLen*)

Compute crc32 according to IEEE802.3.

Calculate CRC for the given buffer and length.

Parameters:

← ***crc*** Initial value.

↔ ***pData*** Pointer to data.

← ***dataLen*** length in bytes of data.

Return values:

crc32 according to IEEE802.3

5.28.2.2 VOS_ERR_T vos_init (void * *pRefCon*, VOS_PRINT_DBG_T *pDebugOutput*)

Initialize the vos library.

This is used to set the output function for all VOS error and debug output.

Parameters:

← ∗***pRefCon*** user context

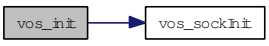
← ∗***pDebugOutput*** pointer to debug output function

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR unsupported

Here is the call graph for this function:

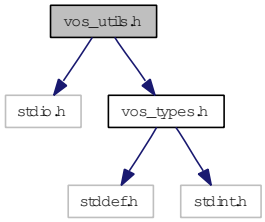


5.29 vos_utils.h File Reference

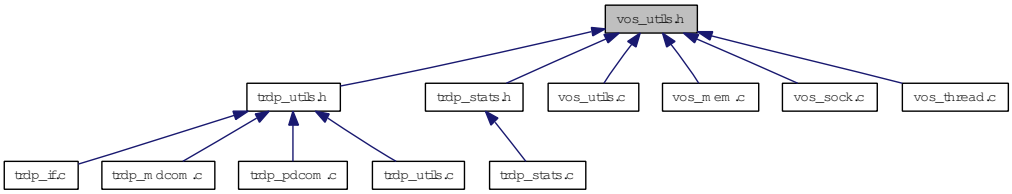
Typedefs for OS abstraction.

```
#include <stdio.h>
#include "vos_types.h"
```

Include dependency graph for vos_utils.h:



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



Defines

- #define vos_print(level, string)

  Debug output macro without formatting options.

- #define vos_printf(level, format, args...)

  Debug output macro with formatting options.

Functions

- EXT_DECL UINT32 vos_crc32 (UINT32 crc, const UINT8 *pData, UINT32 dataLen)

  Calculate CRC for the given buffer and length.

5.29.1 Detailed Description

Typedefs for OS abstraction.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

Id

vos_utils.h 2 2012-06-04 11:25:16Z 97025

5.29.2 Function Documentation

5.29.2.1 EXT_DECL UINT32 vos_crc32 (UINT32 *crc*, const UINT8 * *pData*, UINT32 *dataLen*)

Calculate CRC for the given buffer and length.

For TRDP FCS CRC calculation the CRC32 according to IEEE802.3 with start value 0xffffffff is used.

Parameters:

← ***crc*** Initial value.

↔ ***pData*** Pointer to data.

← ***dataLen*** length in bytes of data.

Return values:

crc32 according to IEEE802.3

Calculate CRC for the given buffer and length.

Parameters:

← ***crc*** Initial value.

↔ ***pData*** Pointer to data.

← ***dataLen*** length in bytes of data.

Return values:

crc32 according to IEEE802.3

Index