

TCNOpen TRDP
PrototypeV1.0

Generated by Doxygen 1.5.6

Fri Jun 14 10:09:10 2013

Contents

1	The TRDP Light Library API Specification	1
1.1	General Information	1
1.1.1	Purpose	1
1.1.2	Scope	1
1.1.3	Related documents	1
1.1.4	Abbreviations and Definitions	1
1.2	Terminology	2
1.3	Conventions of the API	4
2	Data Structure Index	5
2.1	Data Structures	5
3	File Index	7
3.1	File List	7
4	Data Structure Documentation	9
4.1	GNU_PACKED Struct Reference	9
4.1.1	Detailed Description	10
4.1.2	Field Documentation	10
4.1.2.1	protocolVersion	10
4.1.2.2	msgType	10
4.1.2.3	datasetLength	11
4.2	MD_ELE Struct Reference	12
4.2.1	Detailed Description	14
4.2.2	Field Documentation	14
4.2.2.1	pPacket	14
4.3	MD_LIS_ELE Struct Reference	15
4.3.1	Detailed Description	15
4.4	PD_ELE Struct Reference	16

4.4.1	Detailed Description	17
4.4.2	Field Documentation	17
4.4.2.1	pFrame	17
4.5	TAU_MARSHALL_INFO_T Struct Reference	19
4.5.1	Detailed Description	19
4.6	TRDP_CAR_INFO_T Struct Reference	20
4.6.1	Detailed Description	21
4.6.2	Field Documentation	21
4.6.2.1	orient	21
4.6.2.2	pDevInfo	21
4.7	TRDP_COMID_DSID_MAP_T Struct Reference	22
4.7.1	Detailed Description	22
4.8	TRDP_CST_INFO_T Struct Reference	23
4.8.1	Detailed Description	24
4.8.2	Field Documentation	24
4.8.2.1	owner	24
4.8.2.2	orient	24
4.8.2.3	pFctInfo	24
4.8.2.4	pCarInfo	24
4.9	TRDP_DATASET Struct Reference	25
4.9.1	Detailed Description	25
4.10	TRDP_DATASET_ELEMENT_T Struct Reference	26
4.10.1	Detailed Description	26
4.10.2	Field Documentation	26
4.10.2.1	type	26
4.11	TRDP_DBG_CONFIG_T Struct Reference	27
4.11.1	Detailed Description	27
4.12	TRDP_DEVICE_INFO_T Struct Reference	28
4.12.1	Detailed Description	29
4.12.2	Field Documentation	29
4.12.2.1	orient	29
4.13	TRDP_FCT_INFO_T Struct Reference	30
4.13.1	Detailed Description	30
4.14	TRDP_HANDLE Struct Reference	31
4.14.1	Detailed Description	31
4.15	TRDP_LIST_STATISTICS_T Struct Reference	32

4.15.1 Detailed Description	32
4.16 TRDP_MARSHALL_CONFIG_T Struct Reference	33
4.16.1 Detailed Description	33
4.17 TRDP_MD_CONFIG_T Struct Reference	34
4.17.1 Detailed Description	35
4.18 TRDP_MD_INFO_T Struct Reference	36
4.18.1 Detailed Description	37
4.18.2 Field Documentation	37
4.18.2.1 msgType	37
4.19 TRDP_MD_STATISTICS_T Struct Reference	38
4.19.1 Detailed Description	39
4.20 TRDP_MD_TCP Struct Reference	40
4.20.1 Detailed Description	40
4.21 TRDP_MEM_CONFIG_T Struct Reference	41
4.21.1 Detailed Description	41
4.22 TRDP_MEM_STATISTICS_T Struct Reference	42
4.22.1 Detailed Description	42
4.23 TRDP_PD_CONFIG_T Struct Reference	43
4.23.1 Detailed Description	43
4.24 TRDP_PD_INFO_T Struct Reference	44
4.24.1 Detailed Description	44
4.24.2 Field Documentation	45
4.24.2.1 msgType	45
4.25 TRDP_PD_STATISTICS_T Struct Reference	46
4.25.1 Detailed Description	47
4.26 TRDP_PROCESS_CONFIG_T Struct Reference	48
4.26.1 Detailed Description	48
4.27 TRDP_PROP_INFO_T Struct Reference	49
4.27.1 Detailed Description	49
4.28 TRDP_PUB_STATISTICS_T Struct Reference	50
4.28.1 Detailed Description	50
4.28.2 Field Documentation	50
4.28.2.1 destAddr	50
4.29 TRDP_RED_STATISTICS_T Struct Reference	51
4.29.1 Detailed Description	51
4.30 TRDP_SDT_PAR_T Struct Reference	52

4.30.1 Detailed Description	52
4.31 TRDP_SEND_PARAM_T Struct Reference	53
4.31.1 Detailed Description	53
4.32 TRDP_SESSION Struct Reference	54
4.32.1 Detailed Description	55
4.33 TRDP_SOCKET_TCP Struct Reference	56
4.33.1 Detailed Description	56
4.34 TRDP_SOCKETS Struct Reference	57
4.34.1 Detailed Description	57
4.34.2 Field Documentation	58
4.34.2.1 usage	58
4.35 TRDP_STATISTICS_T Struct Reference	59
4.35.1 Detailed Description	60
4.36 TRDP_SUBS_STATISTICS_T Struct Reference	61
4.36.1 Detailed Description	61
4.36.2 Field Documentation	61
4.36.2.1 filterAddr	61
4.36.2.2 timeout	61
4.36.2.3 toBehav	62
4.36.2.4 numRecv	62
4.37 TRDP_TCP_FD_T Struct Reference	63
4.37.1 Detailed Description	63
4.38 TRDP_TRAIN_INFO_T Struct Reference	64
4.38.1 Detailed Description	65
4.38.2 Field Documentation	65
4.38.2.1 operator	65
4.38.2.2 topoCnt	65
4.38.2.3 pCstInfo	65
4.39 TRDP_VERSION_T Struct Reference	66
4.39.1 Detailed Description	66
4.40 TRDP_XML_DOC_HANDLE_T Struct Reference	67
4.40.1 Detailed Description	67
4.41 VOS SOCK_OPT_T Struct Reference	68
4.41.1 Detailed Description	68
4.41.2 Field Documentation	68
4.41.2.1 qos	68

4.42	VOS_TIME_T Struct Reference	69
4.42.1	Detailed Description	69
4.42.2	Field Documentation	69
4.42.2.1	tv_usec	69
5	File Documentation	71
5.1	tau_addr.h File Reference	71
5.1.1	Detailed Description	73
5.1.2	Function Documentation	73
5.1.2.1	tau_addr2CarId	73
5.1.2.2	tau_addr2CarNo	74
5.1.2.3	tau_addr2CstId	74
5.1.2.4	tau_addr2CstNo	74
5.1.2.5	tau_addr2IecCarNo	75
5.1.2.6	tau_addr2IecCstNo	75
5.1.2.7	tau_addr2Uri	75
5.1.2.8	tau_carNo2Ids	76
5.1.2.9	tau_cstNo2CstId	76
5.1.2.10	tau_getOwnAddr	76
5.1.2.11	tau_getOwnIds	77
5.1.2.12	tau_iecCarNo2Ids	77
5.1.2.13	tau_iecCstNo2CstId	77
5.1.2.14	tau_label2CarId	78
5.1.2.15	tau_label2CarNo	78
5.1.2.16	tau_label2CstId	78
5.1.2.17	tau_label2CstNo	79
5.1.2.18	tau_label2IecCarNo	79
5.1.2.19	tau_label2IecCstNo	79
5.1.2.20	tau_uri2Addr	80
5.2	tau_marshall.c File Reference	81
5.2.1	Detailed Description	82
5.2.2	Function Documentation	82
5.2.2.1	tau_calcDatasetSize	82
5.2.2.2	tau_calcDatasetSizeByComId	83
5.2.2.3	tau_initMarshall	83
5.2.2.4	tau_marshall	84
5.2.2.5	tau_marshallIDs	84

5.2.2.6	tau_unmarshall	85
5.2.2.7	tau_unmarshallDs	85
5.3	tau_marshall.h File Reference	86
5.3.1	Detailed Description	87
5.3.2	Function Documentation	87
5.3.2.1	tau_calcDatasetSize	87
5.3.2.2	tau_calcDatasetSizeByComId	88
5.3.2.3	tau_initMarshall	88
5.3.2.4	tau_marshall	89
5.3.2.5	tau_marshallDs	90
5.3.2.6	tau_unmarshall	90
5.3.2.7	tau_unmarshallDs	91
5.4	tau_tti.h File Reference	92
5.4.1	Detailed Description	94
5.4.2	Enumeration Type Documentation	94
5.4.2.1	TRDP_FCT_T	94
5.4.2.2	TRDP_INAUG_STATE_T	95
5.4.3	Function Documentation	95
5.4.3.1	tau_getCarDevCnt	95
5.4.3.2	tau_getCarInfo	95
5.4.3.3	tau_getCarOrient	96
5.4.3.4	tau_getCstCarCnt	96
5.4.3.5	tau_getCstFctCnt	97
5.4.3.6	tau_getCstFctInfo	97
5.4.3.7	tau_getCstInfo	97
5.4.3.8	tau_getDevInfo	98
5.4.3.9	tau_getEtbState	98
5.4.3.10	tau_getIecCarOrient	98
5.4.3.11	tau_getTrnCarCnt	99
5.4.3.12	tau_getTrnCstCnt	99
5.4.3.13	tau_getTrnInfo	99
5.5	tau_xml.c File Reference	100
5.5.1	Detailed Description	101
5.5.2	Define Documentation	102
5.5.2.1	TRDP_SDT_DEFAULT_CMTHR	102
5.5.3	Function Documentation	102

5.5.3.1	tau_freeTelegrams	102
5.5.3.2	tau_freeXmlDoc	102
5.5.3.3	tau_prepareXmlDoc	102
5.5.3.4	tau_readXmlDatasetConfig	103
5.5.3.5	tau_readXmlDeviceConfig	103
5.5.3.6	tau_readXmlInterfaceConfig	103
5.6	tau_xml.h File Reference	105
5.6.1	Detailed Description	106
5.6.2	Enumeration Type Documentation	107
5.6.2.1	TRDP_DBG_OPTION_T	107
5.6.3	Function Documentation	107
5.6.3.1	tau_freeTelegrams	107
5.6.3.2	tau_freeXmlDoc	108
5.6.3.3	tau_prepareXmlDoc	108
5.6.3.4	tau_readXmlDatasetConfig	108
5.6.3.5	tau_readXmlDeviceConfig	109
5.6.3.6	tau_readXmlInterfaceConfig	109
5.7	trdp_dllmain.c File Reference	111
5.7.1	Detailed Description	111
5.8	trdp_if.c File Reference	112
5.8.1	Detailed Description	114
5.8.2	Function Documentation	115
5.8.2.1	tlc_closeSession	115
5.8.2.2	tlc_getInterval	115
5.8.2.3	tlc_getVersion	116
5.8.2.4	tlc_getVersionString	116
5.8.2.5	tlc_init	116
5.8.2.6	tlc_openSession	117
5.8.2.7	tlc_process	119
5.8.2.8	tlc_reinitSession	121
5.8.2.9	tlc_setTopoCount	121
5.8.2.10	tlc_terminate	122
5.8.2.11	tlp_get	122
5.8.2.12	tlp_getRedundant	123
5.8.2.13	tlp_publish	124
5.8.2.14	tlp_put	126

5.8.2.15	tlp_request	127
5.8.2.16	tlp_setRedundant	129
5.8.2.17	tlp_subscribe	129
5.8.2.18	tlp_unpublish	130
5.8.2.19	tlp_unsubscribe	131
5.8.2.20	trdp_isValidSession	132
5.8.2.21	trdp_sessionQueue	132
5.9	trdp_if.h File Reference	133
5.9.1	Detailed Description	133
5.9.2	Function Documentation	134
5.9.2.1	trdp_isValidSession	134
5.9.2.2	trdp_sessionQueue	134
5.10	trdp_if_light.h File Reference	135
5.10.1	Detailed Description	139
5.10.2	Function Documentation	139
5.10.2.1	tlc_closeSession	139
5.10.2.2	tlc_freeBuf	140
5.10.2.3	tlc_getInterval	140
5.10.2.4	tlc_getJoinStatistics	141
5.10.2.5	tlc_getListStatistics	142
5.10.2.6	tlc_getPubStatistics	143
5.10.2.7	tlc_getRedStatistics	144
5.10.2.8	tlc_getStatistics	145
5.10.2.9	tlc_getSubsStatistics	145
5.10.2.10	tlc_getVersion	146
5.10.2.11	tlc_getVersionString	147
5.10.2.12	tlc_init	147
5.10.2.13	tlc_openSession	148
5.10.2.14	tlc_process	150
5.10.2.15	tlc_reinitSession	152
5.10.2.16	tlc_resetStatistics	152
5.10.2.17	tlc_setTopoCount	153
5.10.2.18	tlc_terminate	154
5.10.2.19	tlm_abortSession	154
5.10.2.20	tlm_addListener	155
5.10.2.21	tlm_confirm	155

5.10.2.22	tlm_delListener	156
5.10.2.23	tlm_notify	156
5.10.2.24	tlm_reply	157
5.10.2.25	tlm_replyErr	158
5.10.2.26	tlm_replyQuery	158
5.10.2.27	tlm_request	159
5.10.2.28	tlp_get	160
5.10.2.29	tlp_getRedundant	161
5.10.2.30	tlp_publish	162
5.10.2.31	tlp_put	164
5.10.2.32	tlp_request	166
5.10.2.33	tlp_setRedundant	168
5.10.2.34	tlp_subscribe	168
5.10.2.35	tlp_unpublish	170
5.10.2.36	tlp_unsubscribe	171
5.11	trdp_mdcom.c File Reference	173
5.11.1	Detailed Description	174
5.11.2	Function Documentation	175
5.11.2.1	trdp_closeMDSessions	175
5.11.2.2	trdp_getTCPSocket	175
5.11.2.3	trdp_mdCheck	176
5.11.2.4	trdp_mdCheckListenSocks	176
5.11.2.5	trdp_mdCheckPending	177
5.11.2.6	trdp_mdCheckTimeouts	178
5.11.2.7	trdp_mdFreeSession	178
5.11.2.8	trdp_mdRecv	178
5.11.2.9	trdp_mdRecvPacket	179
5.11.2.10	trdp_mdSend	180
5.11.2.11	trdp_mdSendPacket	181
5.11.2.12	trdp_mdSetSessionTimeout	181
5.11.2.13	trdp_mdUpdatePacket	181
5.12	trdp_mdcom.h File Reference	182
5.12.1	Detailed Description	183
5.12.2	Function Documentation	183
5.12.2.1	trdp_closeMDSessions	183
5.12.2.2	trdp_getTCPSocket	184

5.12.2.3	trdp_mdCheckListenSocks	184
5.12.2.4	trdp_mdCheckPending	185
5.12.2.5	trdp_mdCheckTimeouts	186
5.12.2.6	trdp_mdFreeSession	186
5.12.2.7	trdp_mdRecv	186
5.12.2.8	trdp_mdSend	187
5.12.2.9	trdp_mdSendPacket	188
5.12.2.10	trdp_mdSetSessionTimeout	188
5.12.2.11	trdp_mdUpdatePacket	189
5.13	trdp_pdcom.c File Reference	190
5.13.1	Detailed Description	191
5.13.2	Function Documentation	192
5.13.2.1	trdp_pdCheck	192
5.13.2.2	trdp_pdCheckListenSocks	192
5.13.2.3	trdp_pdCheckPending	193
5.13.2.4	trdp_pdDataUpdate	193
5.13.2.5	trdp_pdDistribute	194
5.13.2.6	trdp_pdHandleTimeOuts	194
5.13.2.7	trdp_pdInit	194
5.13.2.8	trdp_pdReceive	195
5.13.2.9	trdp_pdSend	196
5.13.2.10	trdp_pdSendQueued	197
5.13.2.11	trdp_pdUpdate	197
5.14	trdp_pdcom.h File Reference	198
5.14.1	Detailed Description	199
5.14.2	Function Documentation	200
5.14.2.1	trdp_pdCheck	200
5.14.2.2	trdp_pdCheckListenSocks	200
5.14.2.3	trdp_pdCheckPending	201
5.14.2.4	trdp_pdDataUpdate	201
5.14.2.5	trdp_pdDistribute	202
5.14.2.6	trdp_pdHandleTimeOuts	202
5.14.2.7	trdp_pdInit	202
5.14.2.8	trdp_pdReceive	203
5.14.2.9	trdp_pdSend	204
5.14.2.10	trdp_pdSendQueued	205

5.14.2.11	trdp_pdUpdate	205
5.15	trdp_private.h File Reference	206
5.15.1	Detailed Description	209
5.15.2	Enumeration Type Documentation	209
5.15.2.1	TRDP_MD_ELE_ST_T	209
5.15.2.2	TRDP_PRIV_FLAGS_T	210
5.15.2.3	TRDP SOCK_TYPE_T	210
5.16	trdp_proto.h File Reference	211
5.16.1	Detailed Description	213
5.16.2	Define Documentation	213
5.16.2.1	TRDP_COMID_ECHO	213
5.16.2.2	TRDP_DEST_URI_SIZE	213
5.16.2.3	TRDP_MAX_FILE_NAME_LEN	213
5.16.2.4	TRDP_MAX_LABEL_LEN	214
5.16.2.5	TRDP_MAX_URI_HOST_LEN	214
5.16.2.6	TRDP_MAX_URI_LEN	214
5.16.2.7	TRDP_MAX_URI_USER_LEN	214
5.16.2.8	TRDP_STATISTICS_REQUEST_DSID	214
5.16.3	Enumeration Type Documentation	214
5.16.3.1	TRDP_MSG_T	214
5.17	trdp_stats.c File Reference	215
5.17.1	Detailed Description	216
5.17.2	Function Documentation	216
5.17.2.1	tlc_getJoinStatistics	216
5.17.2.2	tlc_getListStatistics	217
5.17.2.3	tlc_getPubStatistics	217
5.17.2.4	tlc_getRedStatistics	218
5.17.2.5	tlc_getStatistics	219
5.17.2.6	tlc_getSubsStatistics	219
5.17.2.7	tlc_resetStatistics	220
5.17.2.8	trdp_initStats	220
5.17.2.9	trdp_pdPrepareStats	221
5.17.2.10	trdp_UpdateStats	221
5.18	trdp_stats.h File Reference	222
5.18.1	Detailed Description	222
5.18.2	Function Documentation	223

5.18.2.1	trdp_initStats	223
5.18.2.2	trdp_pdPrepareStats	223
5.19	trdp_types.h File Reference	224
5.19.1	Detailed Description	229
5.19.2	Typedef Documentation	229
5.19.2.1	TRDP_IP_ADDR_T	229
5.19.2.2	TRDP_MARSHALL_T	229
5.19.2.3	TRDP_MD_CALLBACK_T	230
5.19.2.4	TRDP_PD_CALLBACK_T	230
5.19.2.5	TRDP_PRINT_DBG_T	230
5.19.2.6	TRDP_TIME_T	230
5.19.2.7	TRDP_UNMARSHALL_T	230
5.19.3	Enumeration Type Documentation	231
5.19.3.1	TRDP_DATA_TYPE_T	231
5.19.3.2	TRDP_ERR_T	231
5.19.3.3	TRDP_FLAGS_T	232
5.19.3.4	TRDP_OPTION_T	233
5.19.3.5	TRDP_RED_STATE_T	233
5.19.3.6	TRDP_REPLY_STATUS_T	233
5.19.3.7	TRDP_TO_BEHAVIOR_T	233
5.20	trdp_utils.c File Reference	234
5.20.1	Detailed Description	236
5.20.2	Function Documentation	236
5.20.2.1	am_big_endian	236
5.20.2.2	trdp_getSeqCnt	236
5.20.2.3	trdp_initSockets	237
5.20.2.4	trdp_isAddressed	237
5.20.2.5	trdp_isRcvSeqCnt	237
5.20.2.6	trdp_MDqueueAppLast	238
5.20.2.7	trdp_MDqueueDelElement	238
5.20.2.8	trdp_MDqueueFindAddr	238
5.20.2.9	trdp_MDqueueInsFirst	239
5.20.2.10	trdp_packetSizeMD	239
5.20.2.11	trdp_packetSizePD	239
5.20.2.12	trdp_queueAppLast	239
5.20.2.13	trdp_queueDelElement	239

5.20.2.14	trdp_queueFindComId	240
5.20.2.15	trdp_queueFindPubAddr	240
5.20.2.16	trdp_queueFindSubAddr	240
5.20.2.17	trdp_queueInsFirst	240
5.20.2.18	trdp_releaseSocket	241
5.20.2.19	trdp_requestSocket	241
5.20.2.20	trdp_SockAddJoin	242
5.20.2.21	trdp_SockDelJoin	242
5.20.2.22	trdp_SockIsJoined	243
5.21	trdp_utils.h File Reference	244
5.21.1	Detailed Description	246
5.21.2	Function Documentation	246
5.21.2.1	am_big_endian	246
5.21.2.2	trdp_getSeqCnt	246
5.21.2.3	trdp_initSockets	247
5.21.2.4	trdp_initUncompletedTCP	247
5.21.2.5	trdp_isAddressed	247
5.21.2.6	trdp_isRcvSeqCnt	247
5.21.2.7	trdp_MDqueueAppLast	248
5.21.2.8	trdp_MDqueueDelElement	248
5.21.2.9	trdp_MDqueueFindAddr	248
5.21.2.10	trdp_MDqueueInsFirst	248
5.21.2.11	trdp_packetSizeMD	249
5.21.2.12	trdp_packetSizePD	249
5.21.2.13	trdp_queueAppLast	249
5.21.2.14	trdp_queueDelElement	249
5.21.2.15	trdp_queueFindComId	249
5.21.2.16	trdp_queueFindPubAddr	250
5.21.2.17	trdp_queueFindSubAddr	250
5.21.2.18	trdp_queueInsFirst	250
5.21.2.19	trdp_releaseSocket	250
5.21.2.20	trdp_requestSocket	251
5.22	vos_mem.c File Reference	253
5.22.1	Detailed Description	254
5.22.2	Function Documentation	255
5.22.2.1	vos_bsearch	255

5.22.2.2	<code>vos_memAlloc</code>	255
5.22.2.3	<code>vos_memCount</code>	256
5.22.2.4	<code>vos_memDelete</code>	256
5.22.2.5	<code>vos_memFree</code>	256
5.22.2.6	<code>vos_memInit</code>	257
5.22.2.7	<code>vos_mutexLocalCreate</code>	257
5.22.2.8	<code>vos_mutexLocalDelete</code>	258
5.22.2.9	<code>vos_qsort</code>	258
5.22.2.10	<code>vos_queueCreate</code>	258
5.22.2.11	<code>vos_queueDestroy</code>	259
5.22.2.12	<code>vos_queueReceive</code>	259
5.22.2.13	<code>vos_queueSend</code>	260
5.22.2.14	<code>vos_strncpy</code>	261
5.22.2.15	<code>vos_strncmp</code>	261
5.23	<code>vos_mem.h</code> File Reference	262
5.23.1	Detailed Description	264
5.23.2	Define Documentation	264
5.23.2.1	<code>VOS_MEM_BLOCKSIZE</code>	264
5.23.2.2	<code>VOS_MEM_PREALLOCATE</code>	264
5.23.3	Function Documentation	264
5.23.3.1	<code>vos_bsearch</code>	264
5.23.3.2	<code>vos_memAlloc</code>	265
5.23.3.3	<code>vos_memCount</code>	265
5.23.3.4	<code>vos_memDelete</code>	266
5.23.3.5	<code>vos_memFree</code>	266
5.23.3.6	<code>vos_memInit</code>	266
5.23.3.7	<code>vos_qsort</code>	267
5.23.3.8	<code>vos_queueCreate</code>	268
5.23.3.9	<code>vos_queueDestroy</code>	268
5.23.3.10	<code>vos_queueReceive</code>	269
5.23.3.11	<code>vos_queueSend</code>	270
5.23.3.12	<code>vos_strncpy</code>	271
5.23.3.13	<code>vos_strncmp</code>	271
5.24	<code>vos_private.h</code> File Reference	272
5.24.1	Detailed Description	272
5.24.2	Function Documentation	273

5.24.2.1	vos_mutexLocalCreate	273
5.24.2.2	vos_mutexLocalDelete	273
5.25	vos_private.h File Reference	274
5.25.1	Detailed Description	274
5.25.2	Function Documentation	275
5.25.2.1	vos_mutexLocalCreate	275
5.25.2.2	vos_mutexLocalDelete	275
5.26	vos_shared_mem.c File Reference	276
5.26.1	Detailed Description	277
5.26.2	Function Documentation	277
5.26.2.1	vos_sharedClose	277
5.26.2.2	vos_sharedOpen	277
5.27	vos_shared_mem.c File Reference	279
5.27.1	Detailed Description	279
5.27.2	Function Documentation	280
5.27.2.1	vos_sharedClose	280
5.27.2.2	vos_sharedOpen	280
5.28	vos_shared_mem.h File Reference	282
5.28.1	Detailed Description	282
5.28.2	Function Documentation	283
5.28.2.1	vos_sharedClose	283
5.28.2.2	vos_sharedOpen	283
5.29	vos_sock.c File Reference	285
5.29.1	Detailed Description	287
5.29.2	Function Documentation	288
5.29.2.1	vos_dottedIP	288
5.29.2.2	vos_getInterfaces	288
5.29.2.3	vos_getMacAddress	288
5.29.2.4	vos_htonl	289
5.29.2.5	vos_htons	289
5.29.2.6	vos_ipDotted	289
5.29.2.7	vos_isMulticast	289
5.29.2.8	vos_ntohl	290
5.29.2.9	vos_ntohs	290
5.29.2.10	vos_select	290
5.29.2.11	vos_sockAccept	290

5.29.2.12	vos_sockBind	291
5.29.2.13	vos_sockClose	292
5.29.2.14	vos_sockConnect	292
5.29.2.15	vos_sockGetMAC	292
5.29.2.16	vos_sockInit	293
5.29.2.17	vos_sockJoinMC	293
5.29.2.18	vos_sockLeaveMC	293
5.29.2.19	vos_sockListen	294
5.29.2.20	vos_sockOpenTCP	294
5.29.2.21	vos_sockOpenUDP	295
5.29.2.22	vos_sockReceiveTCP	295
5.29.2.23	vos_sockReceiveUDP	296
5.29.2.24	vos_sockSendTCP	297
5.29.2.25	vos_sockSendUDP	297
5.29.2.26	vos_sockSetMulticastIf	298
5.29.2.27	vos_sockSetOptions	298
5.30	vos_sock.c File Reference	299
5.30.1	Detailed Description	301
5.30.2	Function Documentation	302
5.30.2.1	vos_dottedIP	302
5.30.2.2	vos_getInterfaces	302
5.30.2.3	vos_htonl	303
5.30.2.4	vos_htons	303
5.30.2.5	vos_ipDotted	303
5.30.2.6	vos_isMulticast	303
5.30.2.7	vos_ntohl	304
5.30.2.8	vos_ntohs	304
5.30.2.9	vos_select	304
5.30.2.10	vos_sockAccept	304
5.30.2.11	vos_sockBind	305
5.30.2.12	vos_sockClose	306
5.30.2.13	vos_sockConnect	306
5.30.2.14	vos_sockGetMAC	306
5.30.2.15	vos_sockInit	307
5.30.2.16	vos_sockJoinMC	307
5.30.2.17	vos_sockLeaveMC	307

5.30.2.18	vos_sockListen	308
5.30.2.19	vos_sockOpenTCP	308
5.30.2.20	vos_sockOpenUDP	309
5.30.2.21	vos_sockReceiveTCP	309
5.30.2.22	vos_sockReceiveUDP	310
5.30.2.23	vos_sockSendTCP	310
5.30.2.24	vos_sockSendUDP	311
5.30.2.25	vos_sockSetMulticastIf	311
5.30.2.26	vos_sockSetOptions	312
5.31	vos_sock.h File Reference	313
5.31.1	Detailed Description	316
5.31.2	Define Documentation	316
5.31.2.1	VOS_MAX_SOCKET_CNT	316
5.31.3	Function Documentation	316
5.31.3.1	vos_dottedIP	316
5.31.3.2	vos_getInterfaces	317
5.31.3.3	vos_htonl	317
5.31.3.4	vos_htons	318
5.31.3.5	vos_ipDotted	318
5.31.3.6	vos_isMulticast	319
5.31.3.7	vos_ntohl	319
5.31.3.8	vos_ntohs	319
5.31.3.9	vos_select	319
5.31.3.10	vos_sockAccept	320
5.31.3.11	vos_sockBind	321
5.31.3.12	vos_sockClose	322
5.31.3.13	vos_sockConnect	322
5.31.3.14	vos_sockGetMAC	323
5.31.3.15	vos_sockInit	324
5.31.3.16	vos_sockJoinMC	324
5.31.3.17	vos_sockLeaveMC	325
5.31.3.18	vos_sockListen	326
5.31.3.19	vos_sockOpenTCP	327
5.31.3.20	vos_sockOpenUDP	328
5.31.3.21	vos_sockReceiveTCP	329
5.31.3.22	vos_sockReceiveUDP	330

5.31.3.23	vos_sockSendTCP	331
5.31.3.24	vos_sockSendUDP	332
5.31.3.25	vos_sockSetMulticastIf	333
5.31.3.26	vos_sockSetOptions	334
5.32	vos_thread.c File Reference	335
5.32.1	Detailed Description	337
5.32.2	Function Documentation	337
5.32.2.1	cyclicThread	337
5.32.2.2	vos_addTime	338
5.32.2.3	vos_clearTime	338
5.32.2.4	vos_cmpTime	338
5.32.2.5	vos_divTime	338
5.32.2.6	vos_getTime	339
5.32.2.7	vos_getTimeStamp	339
5.32.2.8	vos_getUuid	339
5.32.2.9	vos_mulTime	339
5.32.2.10	vos_mutexCreate	339
5.32.2.11	vos_mutexDelete	340
5.32.2.12	vos_mutexLocalCreate	340
5.32.2.13	vos_mutexLocalDelete	341
5.32.2.14	vos_mutexLock	341
5.32.2.15	vos_mutexTryLock	341
5.32.2.16	vos_mutexUnlock	341
5.32.2.17	vos_semaCreate	342
5.32.2.18	vos_semaDelete	342
5.32.2.19	vos_semaGive	342
5.32.2.20	vos_semaTake	343
5.32.2.21	vos_subTime	343
5.32.2.22	vos_threadCreate	343
5.32.2.23	vos_threadDelay	344
5.32.2.24	vos_threadInit	344
5.32.2.25	vos_threadIsActive	344
5.32.2.26	vos_threadTerminate	345
5.33	vos_thread.c File Reference	346
5.33.1	Detailed Description	348
5.33.2	Function Documentation	348

5.33.2.1	cyclicThread	348
5.33.2.2	vos_addTime	349
5.33.2.3	vos_clearTime	349
5.33.2.4	vos_cmpTime	349
5.33.2.5	vos_divTime	350
5.33.2.6	vos_getFreeThreadHandle	350
5.33.2.7	vos_getTime	350
5.33.2.8	vos_getTimeStamp	350
5.33.2.9	vos_getUuid	350
5.33.2.10	vos_mulTime	351
5.33.2.11	vos_mutexCreate	351
5.33.2.12	vos_mutexDelete	351
5.33.2.13	vos_mutexLocalCreate	352
5.33.2.14	vos_mutexLocalDelete	352
5.33.2.15	vos_mutexLock	352
5.33.2.16	vos_mutexTryLock	352
5.33.2.17	vos_mutexUnlock	353
5.33.2.18	vos_semaCreate	353
5.33.2.19	vos_semaDelete	353
5.33.2.20	vos_semaGive	354
5.33.2.21	vos_semaTake	354
5.33.2.22	vos_subTime	354
5.33.2.23	vos_threadCreate	355
5.33.2.24	vos_threadDelay	355
5.33.2.25	vos_threadInit	356
5.33.2.26	vos_threadIsActive	356
5.33.2.27	vos_threadTerminate	356
5.34	vos_thread.h File Reference	357
5.34.1	Detailed Description	359
5.34.2	Function Documentation	360
5.34.2.1	vos_addTime	360
5.34.2.2	vos_clearTime	360
5.34.2.3	vos_cmpTime	360
5.34.2.4	vos_divTime	361
5.34.2.5	vos_getTime	361
5.34.2.6	vos_getTimeStamp	361

5.34.2.7	vos_getUuid	362
5.34.2.8	vos_mulTime	362
5.34.2.9	vos_mutexCreate	362
5.34.2.10	vos_mutexDelete	363
5.34.2.11	vos_mutexLock	363
5.34.2.12	vos_mutexTryLock	364
5.34.2.13	vos_mutexUnlock	364
5.34.2.14	vos_semaCreate	365
5.34.2.15	vos_semaDelete	366
5.34.2.16	vos_semaGive	366
5.34.2.17	vos_semaTake	366
5.34.2.18	vos_subTime	367
5.34.2.19	vos_threadCreate	367
5.34.2.20	vos_threadDelay	369
5.34.2.21	vos_threadInit	369
5.34.2.22	vos_threadIsActive	370
5.34.2.23	vos_threadTerminate	370
5.35	vos_types.h File Reference	371
5.35.1	Detailed Description	372
5.35.2	Typedef Documentation	373
5.35.2.1	VOS_PRINT_DBG_T	373
5.35.3	Enumeration Type Documentation	373
5.35.3.1	VOS_ERR_T	373
5.35.3.2	VOS_LOG_T	374
5.35.4	Function Documentation	374
5.35.4.1	vos_init	374
5.36	vos_utils.c File Reference	375
5.36.1	Detailed Description	375
5.36.2	Function Documentation	376
5.36.2.1	vos_crc32	376
5.36.2.2	vos_init	376
5.36.2.3	vos_initRuntimeConsts	376
5.36.2.4	vos_isBigEndian	377
5.37	vos_utils.h File Reference	378
5.37.1	Detailed Description	379
5.37.2	Define Documentation	379

5.37.2.1	VOS_MAX_ERR_STR_SIZE	379
5.37.2.2	VOS_MAX_FRMT_SIZE	379
5.37.2.3	VOS_MAX_PRNT_STR_SIZE	379
5.37.3	Function Documentation	379
5.37.3.1	vos_crc32	379

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 within 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.

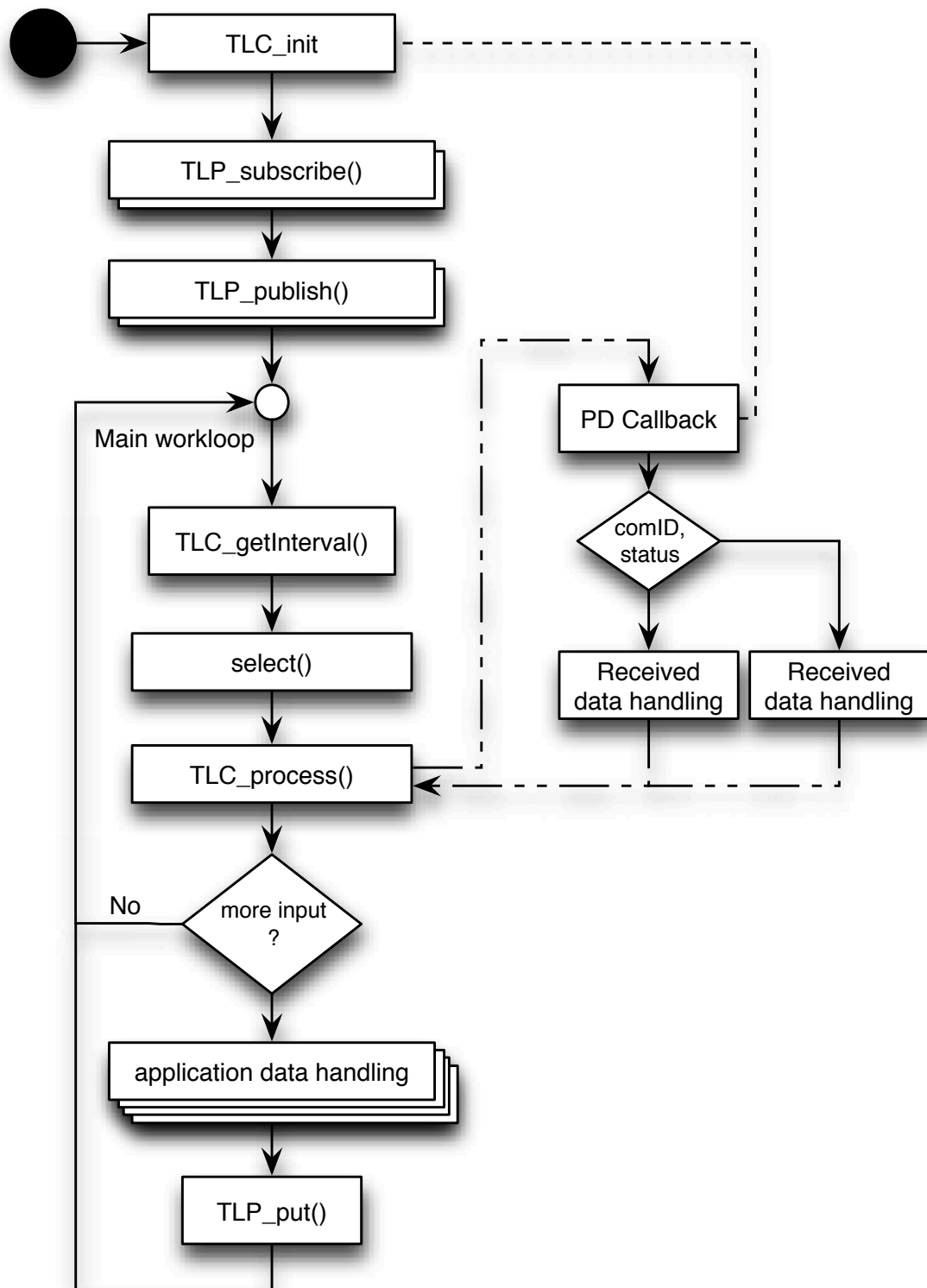


Figure 1.1: Sample client workflow

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:

GNU_PACKED (TRDP process data header - network order and alignment)	9
MD_ELE (Session queue element for MD (UDP and TCP))	12
MD_LIS_ELE (Queue element for MD listeners (UDP and TCP))	15
PD_ELE (Queue element for PD packets to send or receive)	16
TAU_MARSHALL_INFO_T (Marshalling info, used to and from wire)	19
TRDP_CAR_INFO_T (Car information structure)	20
TRDP_COMID_DSID_MAP_T (ComId - data set mapping element definition)	22
TRDP_CST_INFO_T (Consist information structure)	23
TRDP_DATASET (Dataset definition)	25
TRDP_DATASET_ELEMENT_T (Dataset element definition)	26
TRDP_DBG_CONFIG_T (Control for debug output device/file on application level)	27
TRDP_DEVICE_INFO_T (Device information structure)	28
TRDP_FCT_INFO_T (Device information structure)	30
TRDP_HANDLE (Hidden handle definition, used as unique addressing item)	31
TRDP_LIST_STATISTICS_T (Information about a particular MD listener)	32
TRDP_MARSHALL_CONFIG_T (Marshaling/unmarshalling configuration)	33
TRDP_MD_CONFIG_T (Default MD configuration)	34
TRDP_MD_INFO_T (Message data info from received telegram; allows the application to generate responses)	36
TRDP_MD_STATISTICS_T (Structure containing all general MD statistics information)	38
TRDP_MD_TCP (Tcp connection parameters)	40
TRDP_MEM_CONFIG_T (Enumeration type for memory pre-fragmentation, reuse of VOS definition)	41
TRDP_MEM_STATISTICS_T (TRDP statistics type definitions)	42
TRDP_PD_CONFIG_T (Default PD configuration)	43
TRDP_PD_INFO_T (Process data info from received telegram; allows the application to generate responses)	44
TRDP_PD_STATISTICS_T (Structure containing all general PD statistics information)	46
TRDP_PROCESS_CONFIG_T (Various flags/general TRDP options for library initialization) .	48
TRDP_PROP_INFO_T (Properties information structure)	49
TRDP_PUB_STATISTICS_T (Table containing particular PD publishing information)	50
TRDP_RED_STATISTICS_T (A table containing PD redundant group information)	51
TRDP_SDT_PAR_T (Types to read out the XML configuration)	52

TRDP_SEND_PARAM_T (Quality/type of service and time to live)	53
TRDP_SESSION (Session/application variables store)	54
TRDP_SOCKET_TCP (TCP parameters)	56
TRDP_SOCKETS (Socket item)	57
TRDP_STATISTICS_T (Structure containing all general memory, PD and MD statistics information)	59
TRDP_SUBS_STATISTICS_T (Table containing particular PD subscription information)	61
TRDP_TCP_FD_T (TCP file descriptor parameters)	63
TRDP_TRAIN_INFO_T (Train information structure)	64
TRDP_VERSION_T (Version information)	66
TRDP_XML_DOC_HANDLE_T (Parsed XML document handle)	67
VOS SOCK_OPT_T (Common socket options)	68
VOS_TIME_T (Timer value compatible with timeval / select)	69

Chapter 3

File Index

3.1 File List

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

tau_addr.h (TRDP utility interface definitions)	71
tau_marshall.c (Marshalling functions for TRDP)	81
tau_marshall.h (TRDP utility interface definitions)	86
tau_tti.h (TRDP utility interface definitions)	92
tau_xml.c (Functions for XML file parsing)	100
tau_xml.h (TRDP utility interface definitions)	105
trdp_dllmain.c (Windows DLL main function)	111
trdp_if.c (Functions for ECN communication)	112
trdp_if.h (Typedefs for TRDP communication)	133
trdp_if_light.h (TRDP Light interface functions (API))	135
trdp_mdcom.c (Functions for MD communication)	173
trdp_mdcom.h (Functions for MD communication)	182
trdp_pdcom.c (Functions for PD communication)	190
trdp_pdcom.h (Functions for PD communication)	198
trdp_private.h (Typedefs for TRDP communication)	206
trdp_proto.h (Definitions for the TRDP protocol)	211
trdp_stats.c (Statistics functions for TRDP communication)	215
trdp_stats.h (Statistics for TRDP communication)	222
trdp_types.h (Typedefs for TRDP communication)	224
trdp_utils.c (Helper functions for TRDP communication)	234
trdp_utils.h (Common utilities for TRDP communication)	244
vos_mem.c (Memory functions)	253
vos_mem.h (Memory and queue functions for OS abstraction)	262
posix/vos_private.h (Private definitions for the OS abstraction layer)	272
windows/vos_private.h (Private definitions for the OS abstraction layer)	274
posix/vos_shared_mem.c (Shared Memory functions)	276
windows/vos_shared_mem.c (Shared Memory functions)	279
vos_shared_mem.h (Shared Memory functions for OS abstraction)	282
posix/vos_sock.c (Socket functions)	285
windows/vos_sock.c (Socket functions)	299
vos_sock.h (Typedefs for OS abstraction)	313
posix/vos_thread.c (Multitasking functions)	335
windows/vos_thread.c (Multitasking functions)	346

vos_thread.h (Threading functions for OS abstraction)	357
vos_types.h (Typedefs for OS abstraction)	371
vos_utils.c (Common functions for VOS)	375
vos_utils.h (Typedefs for OS abstraction)	378

Chapter 4

Data Structure Documentation

4.1 GNU_PACKED 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 deactivate
- UINT32 [datasetLength](#)
length of the data to transmit 0.
- UINT32 [reserved](#)
before used for ladder support
- UINT32 [replyComId](#)
used in PD request
- UINT32 [replyIpAddress](#)
used for PD request

- UINT32 [frameChecksum](#)
CRC32 of header.
- 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.
- PD_HEADER_T [frameHead](#)
Packet header in network byte order.
- UINT8 [data](#) [TRDP_MAX_PD_PACKET_SIZE]
data ready to be sent or received (with CRCs)
- MD_HEADER_T [frameHead](#)
Packet header in network byte order.

4.1.1 Detailed Description

TRDP process data header - network order and alignment.

TRDP MD packet.

TRDP PD packet.

TRDP message data header - network order and alignment.

4.1.2 Field Documentation

4.1.2.1 UINT16 GNU_PACKED::protocolVersion

fix value for compatibility (set by the API)

fix value for compatibility

4.1.2.2 UINT16 GNU_PACKED::msgType

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

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

4.1.2.3 UINT32 GNU_PACKED::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 files:

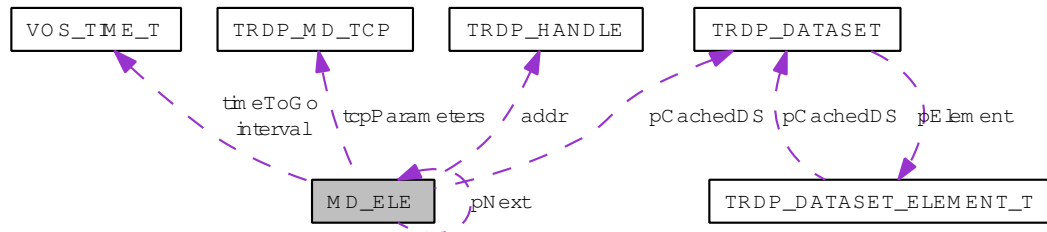
- [trdp_proto.h](#)
- [trdp_private.h](#)

4.2 MD_ELE Struct Reference

Session queue element for MD (UDP and TCP).

```
#include <trdp_private.h>
```

Collaboration diagram for MD_ELE:



Data Fields

- struct [MD_ELE](#) * [pNext](#)
pointer to next element or NULL
- [TRDP_ADDRESSES_T](#) [addr](#)
handle of publisher/subscriber
- [UINT32](#) [curSeqCnt](#)
the last sent or received sequence counter
- [TRDP_PRIV_FLAGS_T](#) [privFlags](#)
private flags
- [TRDP_FLAGS_T](#) [pktFlags](#)
flags
- [BOOL](#) [morituri](#)
about to die
- [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
- [UINT32](#) [dataSize](#)
net data size
- [UINT32](#) [grossSize](#)
complete packet size (header, data, padding, FCS)
- [UINT32](#) [sendSize](#)

data size sent out

- [TRDP_DATASET_T * pCachedDS](#)
Pointer to dataset element if known.
- INT32 [socketIdx](#)
index into the socket list
- UINT16 [replyPort](#)
replies are sent to the requesters source port
- [TRDP_MD_ELE_ST_T stateEle](#)
internal status
- UINT8 [sessionID](#) [16]
UUID as a byte stream.
- UINT32 [numExpReplies](#)
number of expected repliers, 0 if unknown
- UINT32 [numReplies](#)
actual number of replies for the request
- UINT32 [numRetriesMax](#)
maximun number of retries for request to a know dev
- UINT32 [numRetries](#)
actual number of retries for request to a know dev
- UINT32 [numRepliesQuery](#)
number of ReplyQuery received, used to count nuomber of expected Confirm sent
- UINT32 [numConfirmSent](#)
number of Confirm sent
- UINT32 [numConfirmTimeout](#)
number of Confirm Timeouts (incremented by listeners
- const void * [pUserRef](#)
user reference for call_back from [tlm_request\(\)](#)
- [TRDP_URI_USER_T destURI](#)
filter on incoming MD by destination URI
- [TRDP_MD_TCP_T tcpParameters](#)
Tcp connection parameters.
- [MD_PACKET_T * pPacket](#)
Packet header in network byte order.

4.2.1 Detailed Description

Session queue element for MD (UDP and TCP).

4.2.2 Field Documentation

4.2.2.1 MD_PACKET_T* MD_ELE::pPacket

Packet header in network byte order.

data ready to be sent (with CRCs)

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

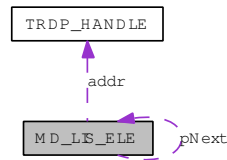
- [trdp_private.h](#)

4.3 MD_LIS_ELE Struct Reference

Queue element for MD listeners (UDP and TCP).

```
#include <trdp_private.h>
```

Collaboration diagram for MD_LIS_ELE:



Data Fields

- struct [MD_LIS_ELE](#) * [pNext](#)
pointer to next element or NULL
- [TRDP_ADDRESSES_T](#) [addr](#)
addressing values
- [TRDP_PRIV_FLAGS_T](#) [privFlags](#)
private flags
- [TRDP_FLAGS_T](#) [pktFlags](#)
flags
- const void * [pUserRef](#)
user reference for call_back from [tlm_request\(\)](#)
- INT32 [socketIdx](#)
index into the socket list

4.3.1 Detailed Description

Queue element for MD listeners (UDP and TCP).

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

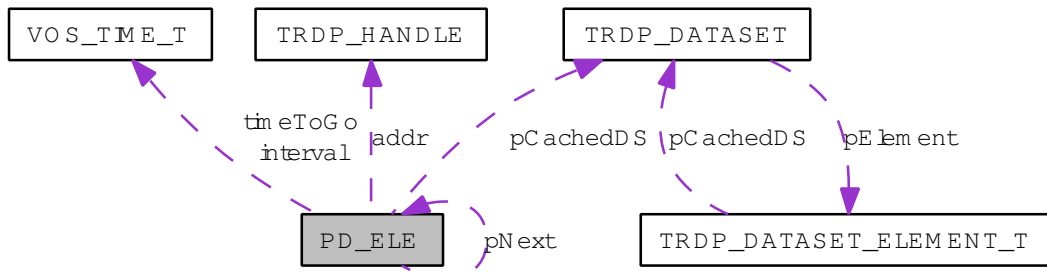
- [trdp_private.h](#)

4.4 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
- **UINT32** **magic**
prevent acces through dangeling pointer
- **TRDP_ADDRESSES_T** **addr**
handle of publisher/subscriber
- **TRDP_IP_ADDR_T** **pullIpAddress**
In case of pulling a PD this is the requested Ip.
- **UINT32** **redId**
Redundancy group ID or zero.
- **UINT32** **curSeqCnt**
the last sent or received sequence counter
- **UINT32** **curSeqCnt4Pull**
the last sent sequence counter for PULL
- **UINT32** **numRxTx**
Counter for received packets (statistics).
- **UINT32** **updPkts**
Counter for updated packets (statistics).
- **UINT32** **getPkts**
Counter for read packets (statistics).

- [TRDP_ERR_T lastErr](#)
Last error (timeout).
- [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
- [TRDP_TO_BEHAVIOR_T toBehavior](#)
timeout behavior for packets
- [UINT32 dataSize](#)
net data size
- [UINT32 grossSize](#)
complete packet size (header, data, padding, FCS)
- [UINT32 sendSize](#)
data size sent out
- [TRDP_DATASET_T * pCachedDS](#)
Pointer to dataset element if known.
- [INT32 socketIdx](#)
index into the socket list
- [const void * userRef](#)
from subscribe()
- [PD_PACKET_T * pFrame](#)
header.

4.4.1 Detailed Description

Queue element for PD packets to send or receive.

4.4.2 Field Documentation

4.4.2.1 PD_PACKET_T* PD_ELE::pFrame

header .

.. data + FCS...

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

- [trdp_private.h](#)

4.5 TAU_MARSHALL_INFO_T Struct Reference

Marshalling info, used to and from wire.

Data Fields

- INT32 [level](#)
track recursive level
- UINT8 * [pSrc](#)
source pointer
- UINT8 * [pDst](#)
destination pointer
- UINT8 * [pDstEnd](#)
last destination

4.5.1 Detailed Description

Marshalling info, used to and from wire.

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

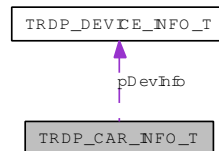
- [tau_marshall.c](#)

4.6 TRDP_CAR_INFO_T Struct Reference

car information structure.

```
#include <tau_tti.h>
```

Collaboration diagram for TRDP_CAR_INFO_T:



Data Fields

- TRDP_LABEL_T [id](#)
Unique car identifier (Label) / IEC identification number.
- TRDP_LABEL_T [type](#)
car type
- UINT8 [orient](#)
0 == opposite, 1 == same orientation rel.
- UINT8 [lead](#)
0 == car is not leading
- UINT8 [leadDir](#)
0 == leading direction 1, 1 == leading direction 2
- UINT8 [no](#)
sequence number of car in consist
- UINT8 [iecNo](#)
IEC sequence number of car in train.
- UINT8 [reachable](#)
0 == car not reachable, inserted manually
- UINT16 [devCnt](#)
number of devices in the car
- TRDP_DEVICE_INFO_T * [pDevInfo](#)
Pointer to device info list for application use and convenience.
- UINT16 [propLen](#)
car property length
- UINT8 * [pProp](#)
Pointer to car properties for application use and convenience.

4.6.1 Detailed Description

car information structure.

4.6.2 Field Documentation

4.6.2.1 `UINT8 TRDP_CAR_INFO_T::orient`

0 == opposite, 1 == same orientation rel.

to consist

4.6.2.2 `TRDP_DEVICE_INFO_T* TRDP_CAR_INFO_T::pDevInfo`

Pointer to device info list for application use and convenience.

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

- [tau_tti.h](#)

4.7 TRDP_COMID_DSID_MAP_T Struct Reference

ComId - data set mapping element definition.

```
#include <trdp_types.h>
```

Data Fields

- UINT32 [comId](#)
comId
- UINT32 [datasetId](#)
corresponding dataset Id

4.7.1 Detailed Description

ComId - data set mapping element definition.

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

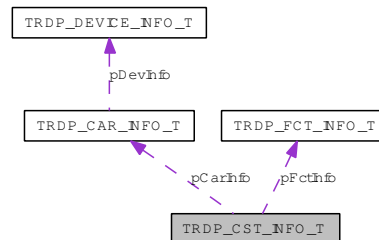
- [trdp_types.h](#)

4.8 TRDP_CST_INFO_T Struct Reference

consist information structure.

```
#include <tau_tti.h>
```

Collaboration diagram for TRDP_CST_INFO_T:



Data Fields

- [TRDP_LABEL_T id](#)
Unique consist identifier (Label) / IEC identification number taken from 1st car in consist.
- [TRDP_LABEL_T owner](#)
consist owner, e.g.
- [TRDP_UUID_T uuid](#)
consist UUID for inauguration purposes
- [UINT8 orient](#)
opposite(0) or same(1) orientation rel.
- [UINT8 lead](#)
0 == consist is not leading
- [UINT8 leadDir](#)
0 == leading direction 1, 1 == leading direction 2
- [UINT8 tcnNo](#)
sequence number of consist in train
- [UINT8 iecNo](#)
IEC sequence number of consist in train.
- [UINT8 reachable](#)
0 == consist not reachable, inserted manually
- [UINT8 ecnCnt](#)
number of cars in the consist
- [UINT8 etbCnt](#)

number of cars in the consist

- `UINT16 fctCnt`
number of public functions in the consist
- `TRDP_FCT_INFO_T * pFctInfo`
Pointer to function info list for application use and convenience.
- `UINT16 carCnt`
number of cars in the consist
- `TRDP_CAR_INFO_T * pCarInfo`
Pointer to car info list for application use and convenience.
- `UINT16 propLen`
consist property length
- `UINT8 * pProp`
Pointer to consist properties for application use and convenience.

4.8.1 Detailed Description

consist information structure.

4.8.2 Field Documentation

4.8.2.1 `TRDP_LABEL_T TRDP_CST_INFO_T::owner`

consist owner, e.g.

"trenitalia.it", "snCF.fr", "db.de"

4.8.2.2 `UINT8 TRDP_CST_INFO_T::orient`

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

to train

4.8.2.3 `TRDP_FCT_INFO_T* TRDP_CST_INFO_T::pFctInfo`

Pointer to function info list for application use and convenience.

4.8.2.4 `TRDP_CAR_INFO_T* TRDP_CST_INFO_T::pCarInfo`

Pointer to car info list for application use and convenience.

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

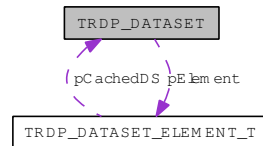
- [tau_tti.h](#)

4.9 TRDP_DATASET Struct Reference

Dataset definition.

```
#include <trdp_types.h>
```

Collaboration diagram for TRDP_DATASET:



Data Fields

- **UINT32 id**
dataset identifier > 1000
- **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.

4.9.1 Detailed Description

Dataset definition.

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

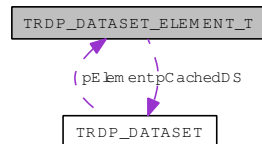
- [trdp_types.h](#)

4.10 TRDP_DATASET_ELEMENT_T Struct Reference

Dataset element definition.

```
#include <trdp_types.h>
```

Collaboration diagram for TRDP_DATASET_ELEMENT_T:



Data Fields

- [UINT32 type](#)
Data type (TRDP_DATA_TYPE_T 1.
- [UINT32 size](#)
Number of items or TDRP_VAR_SIZE (0).
- [struct TRDP_DATASET * pCachedDS](#)
Used internally for marshalling speed-up.

4.10.1 Detailed Description

Dataset element definition.

4.10.2 Field Documentation

4.10.2.1 [UINT32 TRDP_DATASET_ELEMENT_T::type](#)

Data type (TRDP_DATA_TYPE_T 1.

..99) or dataset id > 1000

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

- [trdp_types.h](#)

4.11 TRDP_DBG_CONFIG_T Struct Reference

Control for debug output device/file on application level.

```
#include <tau_xml.h>
```

Data Fields

- [TRDP_DBG_OPTION_T option](#)
Debug printout options for application use.
- `UINT32` [maxFileSize](#)
Maximal file size.
- `TRDP_FILE_NAME_T` [fileName](#)
Debug file name and path.

4.11.1 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](#)

4.12 TRDP_DEVICE_INFO_T Struct Reference

device information structure

```
#include <tau_tti.h>
```

Data Fields

- TRDP_IP_ADDR [addr1](#)
First device IP address.
- TRDP_IP_ADDR [addr2](#)
Second device IP address.
- TRDP_LABEL_T [id](#)
consist unique device identifier (Label) / host name
- TRDP_LABEL_T [type](#)
device type (reserved key words ETBN, ETBR, FCT)
- UINT8 [orient](#)
device orientation 0=opposite, 1=same rel.
- TRDP_LABEL_T [redId](#)
redundant device Id if available
- UINT8 [ecnId1](#)
First consist network id the device is connected to.
- UINT8 [ecnId2](#)
Second consist network id the device is connected to.
- UINT8 [etbId1](#)
First Ethernet train backbone id.
- UINT8 [etbId2](#)
Second Ethernet train backbone id.
- UINT16 [fctCnt](#)
number of public functions on the device
- UINT32 * [pFctNo](#)
Pointer to function number list for application use and convenience.
- UINT16 [propLen](#)
device property length
- UINT8 * [pProp](#)
Pointer to device properties for application use and convenience.

4.12.1 Detailed Description

device information structure

4.12.2 Field Documentation

4.12.2.1 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_tti.h](#)

4.13 TRDP_FCT_INFO_T Struct Reference

device information structure

```
#include <tau_tti.h>
```

Data Fields

- [TRDP_LABEL_T id](#)
function identifier (name)
- [TRDP_FCT_T type](#)
function type
- [UINT32 no](#)
unique function number in consist, should be the list index number
- [TRDP_IP_ADDR addr](#)
Device IP address/multicast address.
- [UINT8 ecnId](#)
Consist network id the device is connected to.
- [UINT8 etbId](#)
Ethernet train backbone id.

4.13.1 Detailed Description

device information structure

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

- [tau_tti.h](#)

4.14 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
- [UINT32 topoCount](#)
topocount belongs to addressing item

4.14.1 Detailed Description

Hidden handle definition, used as unique addressing item.

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

- [trdp_private.h](#)

4.15 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.

4.15.1 Detailed Description

Information about a particular MD listener.

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

- [trdp_types.h](#)

4.16 TRDP_MARSHALL_CONFIG_T Struct Reference

Marshaling/unmarshalling configuration.

```
#include <trdp_types.h>
```

Data Fields

- [TRDP_MARSHALL_T pfCbMarshall](#)
Pointer to marshall callback function.
- [TRDP_UNMARSHALL_T pfCbUnmarshall](#)
Pointer to unmarshall callback function.
- void * [pRefCon](#)
Pointer to user context for call back.

4.16.1 Detailed Description

Marshaling/unmarshalling configuration.

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

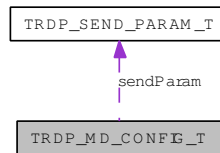
- [trdp_types.h](#)

4.17 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 pfCbFunction](#)
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 reply timeout in us.
- [UINT32 confirmTimeout](#)
Default confirmation timeout in us.
- [UINT32 connectTimeout](#)
Default connection timeout in us.
- [UINT32 sendingTimeout](#)
Default sending timeout in us.
- [UINT16 udpPort](#)
Port to be used for UDP MD communication.
- [UINT16 tcpPort](#)
Port to be used for TCP MD communication.
- [UINT32 maxNumSessions](#)
Maximal number of replier sessions.

4.17.1 Detailed Description

Default MD configuration.

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

- [trdp_types.h](#)

4.18 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
- [BOOL aboutToDie](#)
session is about to die
- [UINT32 numRepliesQuery](#)
number of ReplyQuery received
- [UINT32 numConfirmSent](#)
number of Confirm sent
- [UINT32 numConfirmTimeout](#)
number of Confirm Timeouts (incremented by listeners
- [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 numExpReplies`
number of expected replies, 0 if unknown
- `UINT32 numReplies`
actual number of replies for the request
- `const void * pUserRef`
User reference given with the local call.
- `TRDP_ERR_T resultCode`
error code

4.18.1 Detailed Description

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

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

4.18.2 Field Documentation

4.18.2.1 `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`

4.19 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

4.19.1 Detailed Description

Structure containing all general MD statistics information.

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

- [trdp_types.h](#)

4.20 TRDP_MD_TCP Struct Reference

Tcp connection parameters.

```
#include <trdp_private.h>
```

Data Fields

- BOOL [doConnect](#)
TCP connection state.
- BOOL [msgUncomplete](#)
The receive message is uncomplete.

4.20.1 Detailed Description

Tcp connection parameters.

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

- [trdp_private.h](#)

4.21 TRDP_MEM_CONFIG_T Struct Reference

Enumeration type for memory pre-fragmentation, reuse of VOS definition.

```
#include <trdp_types.h>
```

Data Fields

- `UINT8 * p`
pointer to static or allocated memory
- `UINT32 size`
size of static or allocated memory
- `UINT32 prealloc [VOS_MEM_NBBLOCKSIZES]`
memory block structure

4.21.1 Detailed Description

Enumeration type for memory pre-fragmentation, reuse of VOS definition.

Structure describing memory (and its pre-fragmentation)

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

- [trdp_types.h](#)

4.22 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 [blockSize](#) [VOS_MEM_NBLOCKSIZES]
preallocated memory blocks
- UINT32 [usedBlockSize](#) [VOS_MEM_NBLOCKSIZES]
used memory blocks

4.22.1 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?, timeout, 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:

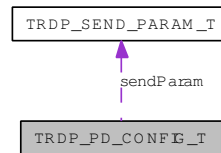
- [trdp_types.h](#)

4.23 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 pfCbFunction](#)
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.
- UINT16 [port](#)
Port to be used for PD communication.

4.23.1 Detailed Description

Default PD configuration.

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

- [trdp_types.h](#)

4.24 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
- [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

4.24.1 Detailed Description

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

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

4.24.2 Field Documentation

4.24.2.1 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](#)

4.25 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

4.25.1 Detailed Description

Structure containing all general PD statistics information.

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

- [trdp_types.h](#)

4.26 TRDP_PROCESS_CONFIG_T Struct Reference

Various flags/general TRDP options for library initialization.

```
#include <trdp_types.h>
```

Data Fields

- `TRDP_LABEL_T` [hostName](#)
Host name.
- `TRDP_LABEL_T` [leaderName](#)
Leader name dependant on redundancy concept.
- `UINT32` [cycleTime](#)
TRDP main process cycle time in us.
- `UINT32` [priority](#)
TRDP main process cycle time (0-255, 0=default, 255=highest).
- `TRDP_OPTION_T` [options](#)
TRDP options.

4.26.1 Detailed Description

Various flags/general TRDP options for library initialization.

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

- [trdp_types.h](#)

4.27 TRDP_PROP_INFO_T Struct Reference

properties information structure

```
#include <tau_tti.h>
```

Data Fields

- `UINT32` [crc](#)
property CRC
- `UINT16` [len](#)
function type
- `UINT8` [ver](#)
property version
- `UINT8` [rel](#)
property release
- `UINT8` [data](#) [1]
dummy field for data access

4.27.1 Detailed Description

properties information structure

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

- [tau_tti.h](#)

4.28 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.

4.28.1 Detailed Description

Table containing particular PD publishing information.

4.28.2 Field Documentation

4.28.2.1 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](#)

4.29 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.

4.29.1 Detailed Description

A table containing PD redundant group information.

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

- [trdp_types.h](#)

4.30 TRDP_SDT_PAR_T Struct Reference

Types to read out the XML configuration.

```
#include <tau_xml.h>
```

Data Fields

- [UINT32 smi1](#)
Safe message identifier - unique for this message at consist level.
- [UINT32 smi2](#)
Safe message identifier - unique for this message at consist level.
- [UINT32 cmThr](#)
Channel monitoring threshold.
- [UINT16 udv](#)
User data version.
- [UINT16 rxPeriod](#)
Sink cycle time.
- [UINT16 txPeriod](#)
Source cycle time.
- [UINT16 nGuard](#)
Initial timeout cycles.
- [UINT8 nrxSafe](#)
Timeout cycles.
- [UINT8 reserved1](#)
Reserved for future use.
- [UINT16 reserved2](#)
Reserved for future use.

4.30.1 Detailed Description

Types to read out the XML configuration.

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

- [tau_xml.h](#)

4.31 TRDP_SEND_PARAM_T Struct Reference

Quality/type of service and time to live.

```
#include <trdp_types.h>
```

Data Fields

- `UINT8 qos`
Quality of service (default should be 5 for PD and 3 for MD).
- `UINT8 ttl`
Time to live (default should be 64).

4.31.1 Detailed Description

Quality/type of service and time to live.

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

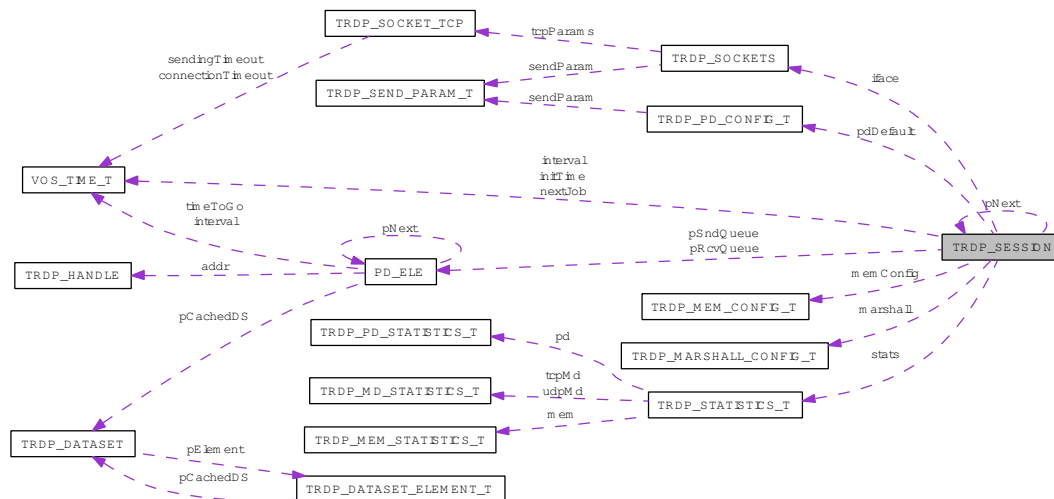
- [trdp_types.h](#)

4.32 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
- [TRDP_TIME_T initTime](#)
initialization time of session
- [TRDP_STATISTICS_T stats](#)
statistics of this session

4.32.1 Detailed Description

Session/application variables store.

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

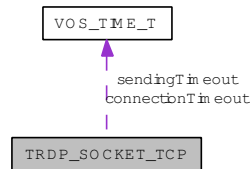
- [trdp_private.h](#)

4.33 TRDP_SOCKET_TCP Struct Reference

TCP parameters.

```
#include <trdp_private.h>
```

Collaboration diagram for TRDP_SOCKET_TCP:



Data Fields

- [TRDP_IP_ADDR_T cornerIp](#)
The other TCP corner Ip.
- [BOOL notSend](#)
If the message has been sent uncompleted.
- [TRDP_TIME_T connectionTimeout](#)
TCP socket connection Timeout.
- [BOOL sendNotOk](#)
The sending timeout will be start.
- [TRDP_TIME_T sendingTimeout](#)
The timeout sending the message.
- [BOOL addFileDesc](#)
Ready to add the socket in the fd.
- [BOOL morituri](#)
about to die

4.33.1 Detailed Description

TCP parameters.

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

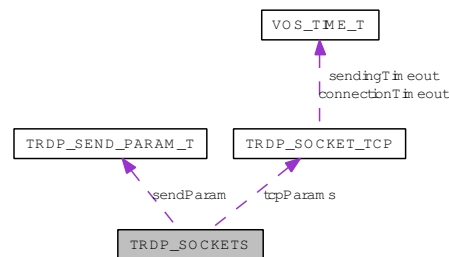
- [trdp_private.h](#)

4.34 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_SOCKET_TYPE_T [type](#)
Usage of this socket.
- BOOL [rcvMostly](#)
Used for receiving.
- INT16 [usage](#)
No.
- TRDP_SOCKET_TCP_T [tcpParams](#)
Params used for TCP.
- TRDP_IP_ADDR_T [mcGroups](#) [VOS_MAX_MULTICAST_CNT]
List of multicast addresses for this socket.

4.34.1 Detailed Description

Socket item.

4.34.2 Field Documentation

4.34.2.1 INT16 TRDP_SOCKETS::usage

No.

of current users of this socket

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

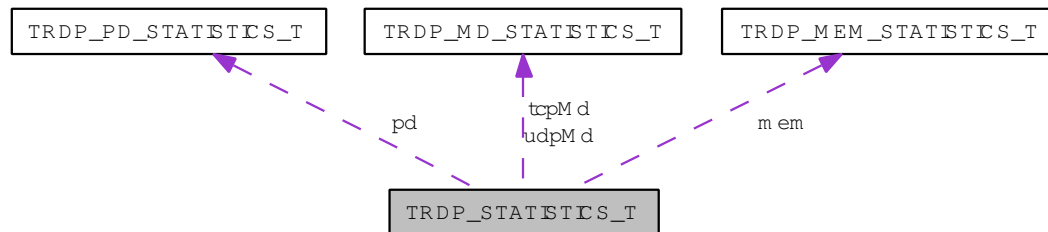
- [trdp_private.h](#)

4.35 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.
- `TIMEDATE64` [timeStamp](#)
actual time stamp
- `TIMEDATE32` [upTime](#)
time in sec since last initialisation
- `TIMEDATE32` [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
- `UINT32` [numJoin](#)

number of joins

- [UINT32 numRed](#)
number of redundancy groups
- [TRDP_MEM_STATISTICS_T mem](#)
memory statistics
- [TRDP_PD_STATISTICS_T pd](#)
pd statistics
- [TRDP_MD_STATISTICS_T udpMd](#)
UDP md statistics.
- [TRDP_MD_STATISTICS_T tcpMd](#)
TCP md statistics.

4.35.1 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](#)

4.36 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.
- [void * 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.

4.36.1 Detailed Description

Table containing particular PD subscription information.

4.36.2 Field Documentation

4.36.2.1 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.

4.36.2.2 UINT32 TRDP_SUBS_STATISTICS_T::timeout

Time-out value in us.

0 = No time-out supervision

4.36.2.3 TRDP_TO_BEHAVIOR_T TRDP_SUBS_STATISTICS_T::toBehav

Behaviour at time-out.

Set data to zero / keep last value

4.36.2.4 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](#)

4.37 TRDP_TCP_FD_T Struct Reference

TCP file descriptor parameters.

```
#include <trdp_private.h>
```

Data Fields

- INT32 [listen_sd](#)
TCP general socket listening connection requests.
- INT32 [max_sd](#)
Maximum socket number in the file descriptor.

4.37.1 Detailed Description

TCP file descriptor parameters.

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

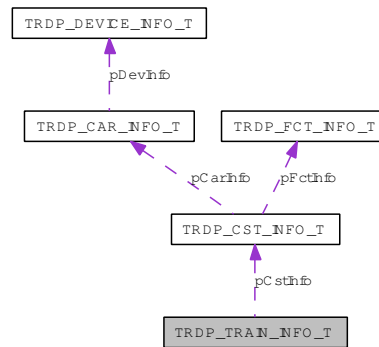
- [trdp_private.h](#)

4.38 TRDP_TRAIN_INFO_T Struct Reference

train information structure.

```
#include <tau_tti.h>
```

Collaboration diagram for TRDP_TRAIN_INFO_T:



Data Fields

- `UINT32` [version](#)
Train info structure version.
- `TRDP_LABEL_T` [id](#)
Train identifier.
- `TRDP_LABEL_T` [operator](#)
Train operator e.g.
- `TRDP_INAUG_STATE_T` [inaugState](#)
inauguration state
- `UINT32` [topoCnt](#)
IEC (i.e.
- `UINT8` [iecOrient](#)
0 == IEC reference orientation is opposite to TCN
- `UINT16` [carCnt](#)
Total number of cars in train.
- `UINT32` [cstCnt](#)
Total number of consists in train.
- `TRDP_CST_INFO_T` * [pCstInfo](#)
Pointer to consist info list for application use and convenience.

4.38.1 Detailed Description

train information structure.

4.38.2 Field Documentation

4.38.2.1 TRDP_LABEL_T TRDP_TRAIN_INFO_T::operator

Train operator e.g.

"trenitalia.it", "snCF.fr", "db.de"

4.38.2.2 UINT32 TRDP_TRAIN_INFO_T::topoCnt

IEC (i.e.

TCN) topography counter

4.38.2.3 TRDP_CST_INFO_T* TRDP_TRAIN_INFO_T::pCstInfo

Pointer to consist info list for application use and convenience.

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

- [tau_tti.h](#)

4.39 TRDP_VERSION_T Struct Reference

Version information.

```
#include <trdp_types.h>
```

Data Fields

- `UINT8 ver`
Version - incremented for incompatible changes.
- `UINT8 rel`
Release - incremented for compatible changes.
- `UINT8 upd`
Update - incremented for bug fixes.
- `UINT8 evo`
Evolution - incremented for build.

4.39.1 Detailed Description

Version information.

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

- [trdp_types.h](#)

4.40 TRDP_XML_DOC_HANDLE_T Struct Reference

Parsed XML document handle.

```
#include <tau_xml.h>
```

Data Fields

- void * [pXmlDocument](#)
Pointer to parsed XML document.
- void * [pRootElement](#)
Pointer to the document root element.
- void * [pXPathContext](#)
Pointer to prepared XPath context.

4.40.1 Detailed Description

Parsed XML document handle.

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

- [tau_xml.h](#)

4.41 VOS_SOCKET_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

4.41.1 Detailed Description

Common socket options.

4.41.2 Field Documentation

4.41.2.1 `UINT8 VOS_SOCKET_OPT_T::qos`

quality/type of service 0.

..7

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

- `vos_sock.h`

4.42 VOS_TIME_T Struct Reference

Timer value compatible with timeval / select.

```
#include <vos_types.h>
```

Data Fields

- UINT32 [tv_sec](#)
full seconds
- INT32 [tv_usec](#)
Micro seconds (max.

4.42.1 Detailed Description

Timer value compatible with timeval / select.

Relative or absolute date, depending on usage

4.42.2 Field Documentation

4.42.2.1 INT32 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

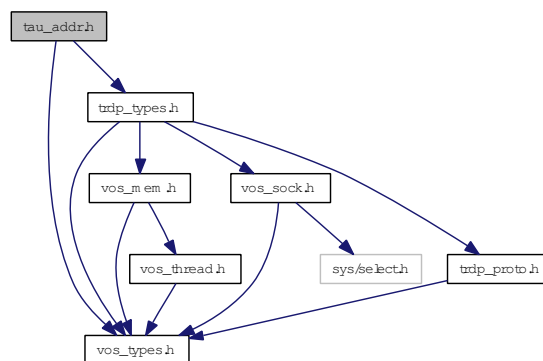
5.1 tau_addr.h File Reference

TRDP utility interface definitions.

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

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

Include dependency graph for tau_addr.h:



Functions

- 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_IP_ADDR tau_getOwnAddr (void)

Function to get the own IP address.

- EXT_DECL TRDP_ERR_T tau_uri2Addr (TRDP_IP_ADDR *pAddr, UINT32 *pTopoCnt, const TRDP_URI_T uri)

Function to convert a URI to an IP address.

- EXT_DECL [TRDP_ERR_T tau_addr2Uri](#) (TRDP_URI_HOST_T uri, UINT32 *pTopoCnt, TRDP_IP_ADDR addr)

Function to convert an IP address to a URL.

- EXT_DECL [TRDP_ERR_T tau_label2CarId](#) (TRDP_LABEL_T carId, UINT32 *pTopoCnt, const TRDP_LABEL_T carLabel, const TRDP_LABEL_T cstLabel)

Function to retrieve the carId of the car with label carLabel in the consist with cstLabel.

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

Function The function delivers the car number to the given label.

- EXT_DECL [TRDP_ERR_T tau_label2IecCarNo](#) (UINT8 *pIecCarNo, UINT32 *pTopoCnt, const TRDP_LABEL_T carLabel, const TRDP_LABEL_T cstLabel)

Function The function delivers the IEC car number to the given label.

- EXT_DECL [TRDP_ERR_T tau_carNo2Ids](#) (TRDP_LABEL_T carId, TRDP_LABEL_T cstId, UINT32 *pTopoCnt, UINT8 carNo, UINT8 trnCstNo)

Function to retrieve the car and consist id of the car given with carNo and trnCstNo.

- EXT_DECL [TRDP_ERR_T tau_iecCarNo2Ids](#) (TRDP_LABEL_T carId, TRDP_LABEL_T cstId, UINT32 *pTopoCnt, UINT8 iecCarNo)

Function to retrieve the car and consist id from a given IEC car sequence number.

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

Function to retrieve the carId of the car hosting a device with the IPAddress ipAddr.

- EXT_DECL [TRDP_ERR_T tau_addr2CarNo](#) (UINT8 *pCarNo, UINT8 *pTopoCnt, TRDP_IP_ADDR ipAddr)

Function to retrieve the car number in consist of the car hosting the device with the IP address ipAddr.

- EXT_DECL [TRDP_ERR_T tau_addr2IecCarNo](#) (UINT8 *pIecCarNo, UINT8 *pTopoCnt, TRDP_IP_ADDR ipAddr)

Function to retrieve the IEC car sequence number of the car hosting the device with the IP address ipAddr.

- EXT_DECL [TRDP_ERR_T tau_cstNo2CstId](#) (TRDP_LABEL_T cstId, UINT32 *pTopoCnt, UINT8 cstNo)

Function to retrieve the consist identifier of the consist with train consist sequence number cstNo.

- EXT_DECL [TRDP_ERR_T tau_iecCstNo2CstId](#) (TRDP_LABEL_T cstId, UINT32 *pTopoCnt, UINT8 iecCstNo)

Function to retrieve the consist identifier of the consist with IEC sequence consist number iecCstNo.

- EXT_DECL [TRDP_ERR_T tau_label2CstId](#) (TRDP_LABEL_T cstId, UINT32 *pTopoCnt, const TRDP_LABEL_T carLabel, const TRDP_LABEL_T cstLabel)

Function to retrieve the consist identifier of the consist hosting a car with label carLabel.

- EXT_DECL [TRDP_ERR_T tau_label2CstNo](#) (UINT8 *pCstNo, UINT32 *pTopoCnt, const TRDP_LABEL_T carLabel)

Function to retrieve the consist sequence number of the consist hosting a car with label carLabel.

- EXT_DECL [TRDP_ERR_T tau_label2IecCstNo](#) (UINT8 *pIecCstNo, UINT32 *pTopoCnt, const TRDP_LABEL_T carLabel)
Function to retrieve the leading car depending IEC consist sequence number of the consist hosting a car with label carLabel.
- EXT_DECL [TRDP_ERR_T tau_addr2CstId](#) (TRDP_LABEL_T cstId, UINT32 *pTopoCnt, TRDP_IP_ADDR ipAddr)
Function to retrieve the consist identifier of the consist hosting the device with the IP-Address ipAddr.
- EXT_DECL [TRDP_ERR_T tau_addr2CstNo](#) (UINT8 *pCstNo, UINT32 *pTopoCnt, TRDP_IP_ADDR ipAddr)
Function to retrieve the consist sequence number of the consist hosting the device with the IP-Address ipAddr.
- EXT_DECL [TRDP_ERR_T tau_addr2IecCstNo](#) (UINT8 *pIecCstNo, UINT32 *pTopoCnt, TRDP_IP_ADDR ipAddr)
Function to retrieve the leading car depending iec consist number of the consist hosting the device with the IP-Address addr.

5.1.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- IP - URI address translation

Note:

Project: TCNOpen TRDP prototype stack

Author:

Armin-H. Weiss (initial version)

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[tau_addr.h](#) 274 2013-01-10 11:00:43Z aweiss

5.1.2 Function Documentation

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

Function to retrieve the carId of the car hosting a device with the IPAddress ipAddr.

Parameters:

- *carId* Pointer to the car id to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *ipAddr* IP address. 0 means own address, so the own car id is returned.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.2 EXT_DECL TRDP_ERR_T tau_addr2CarNo (UINT8 * *pCarNo*, UINT8 * *pTopoCnt*, TRDP_IP_ADDR *ipAddr*)

Function to retrieve the car number in consist of the car hosting the device with the IP address *ipAddr*.

Parameters:

- *pCarNo* Pointer to the car number in consist to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *ipAddr* IP address. 0 means own address, so the own car number is returned.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

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

Function to retrieve the consist identifier of the consist hosting the device with the IP-Address *ipAddr*.

Parameters:

- *cstId* Pointer to the consist id to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *ipAddr* IP address. 0 means own device, so the own consist id is returned.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.4 EXT_DECL TRDP_ERR_T tau_addr2CstNo (UINT8 * *pCstNo*, UINT32 * *pTopoCnt*, TRDP_IP_ADDR *ipAddr*)

Function to retrieve the consist sequence number of the consist hosting the device with the IP-Address *ipAddr*.

Parameters:

- *pCstNo* Pointer to the train consist number to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *ipAddr* IP address. 0 means own device, so the own consist number is returned.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.5 EXT_DECL TRDP_ERR_T tau_addr2IecCarNo (UINT8 * *pIecCarNo*, UINT8 * *pTopoCnt*, TRDP_IP_ADDR *ipAddr*)

Function to retrieve the IEC car sequence number of the car hosting the device with the IP address *ipAddr*.

Parameters:

- *pIecCarNo* Pointer to the IEC car sequence number to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *ipAddr* IP address. 0 means own address, so the own IEC car number is returned.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.6 EXT_DECL TRDP_ERR_T tau_addr2IecCstNo (UINT8 * *pIecCstNo*, UINT32 * *pTopoCnt*, TRDP_IP_ADDR *ipAddr*)

Function to retrieve the leading car depending iec consist number of the consist hosting the device with the IP-Address *addr*.

Parameters:

- *pIecCstNo* Pointer to the iec consist number to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *ipAddr* IP address. 0 means own device, so the own IEC consist number is returned.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.7 EXT_DECL TRDP_ERR_T tau_addr2Uri (TRDP_URI_HOST_T *uri*, UINT32 * *pTopoCnt*, TRDP_IP_ADDR *addr*)

Function to convert an IP address to a URI.

Receives an IP-Address and translates it into the host part of the corresponding URI. Both unicast and multicast addresses are accepted.

Parameters:

- *uri* Pointer to a string to return the URI host part
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *addr* IP address, 0==own address

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.8 EXT_DECL TRDP_ERR_T tau_carNo2Ids (TRDP_LABEL_T *carId*, TRDP_LABEL_T *cstId*, UINT32 **pTopoCnt*, UINT8 *carNo*, UINT8 *trnCstNo*)

Function to retrieve the car and consist id of the car given with *carNo* and *trnCstNo*.

Parameters:

- *carId* Pointer to the car id to be returned
- *cstId* Pointer to the consist id to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carNo* Car number in consist. 0 means own car when *trnCstNo* == 0.
- ← *trnCstNo* Consist sequence number in train. 0 means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.9 EXT_DECL TRDP_ERR_T tau_cstNo2CstId (TRDP_LABEL_T *cstId*, UINT32 **pTopoCnt*, UINT8 *cstNo*)

Function to retrieve the consist identifier of the consist with train consist sequence number *cstNo*.

Parameters:

- *cstId* Pointer to the consist id to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *cstNo* Consist sequence number based on IP reference direction. 0 means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.10 EXT_DECL TRDP_IP_ADDR tau_getOwnAddr (void)

Function to get the own IP address.

Return values:

- own* IP address

5.1.2.11 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

5.1.2.12 EXT_DECL TRDP_ERR_T tau_iecCarNo2Ids (TRDP_LABEL_T *carId*, TRDP_LABEL_T *cstId*, UINT32 **pTopoCnt*, UINT8 *iecCarNo*)

Function to retrieve the car and consist id from a given IEC car sequence number.

Parameters:

- *carId* Pointer to the car id to be returned
- *cstId* Pointer to the consist id to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *iecCarNo* IEC car sequence number. 0 means own car.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.13 EXT_DECL TRDP_ERR_T tau_iecCstNo2CstId (TRDP_LABEL_T *cstId*, UINT32 **pTopoCnt*, UINT8 *iecCstNo*)

Function to retrieve the consist identifier of the consist with IEC sequence consist number *iecCstNo*.

Parameters:

- *cstId* Pointer to the consist id to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *iecCstNo* Consist sequence number based on the leading car depending iec reference direction. 0 means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.14 EXT_DECL TRDP_ERR_T tau_label2CarId (TRDP_LABEL_T *carId*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*)

Function to retrieve the carId of the car with label carLabel in the consist with cstLabel.

Parameters:

- *carId* Pointer to a label string to return the car id
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* Pointer to the car label. NULL means own car if cstLabel == NULL.
- ← *cstLabel* Pointer to the consist label. NULL means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

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

Function 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:

- *pCarNo* Pointer to the car number to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* Pointer to the car label. NULL means own car.
- ← *cstLabel* Pointer to the consist label. NULL means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

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

Function to retrieve the consist identifier of the consist hosting a car with label carLabel.

Parameters:

- *cstId* Pointer to the consist id to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* Pointer to a car label. NULL means any car.
- ← *cstLabel* Pointer to a consist label. NULL means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.17 EXT_DECL TRDP_ERR_T tau_label2CstNo (UINT8 * *pCstNo*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *carLabel*)

Function to retrieve the consist sequence number of the consist hosting a car with label *carLabel*.

Parameters:

- *pCstNo* Pointer to the train consist number to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* Pointer to a car label, NULL means own car, so the own consist number is returned.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.18 EXT_DECL TRDP_ERR_T tau_label2IecCarNo (UINT8 * *pIecCarNo*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*)

Function The function delivers the IEC car number to the given label.

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

Parameters:

- *pIecCarNo* Pointer to the IEC car sequence number to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* Pointer to a car label. NULL means own car.
- ← *cstLabel* Pointer to a consist label. NULL means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.19 EXT_DECL TRDP_ERR_T tau_label2IecCstNo (UINT8 * *pIecCstNo*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *carLabel*)

Function to retrieve the leading car depending IEC consist sequence number of the consist hosting a car with label *carLabel*.

Parameters:

- *pIecCstNo* Pointer to the iec consist number to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* Pointer to a car label. NULL means own car, so the own IEC consist number is returned.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.1.2.20 EXT_DECL TRDP_ERR_T tau_uri2Addr (TRDP_IP_ADDR * *pAddr*, UINT32 * *pTopoCnt*, const TRDP_URI_T *uri*)

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:

- *pAddr* Pointer to return the IP address
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *uri* Pointer to a URI or an IP Address string, NULL==own URI

Return values:

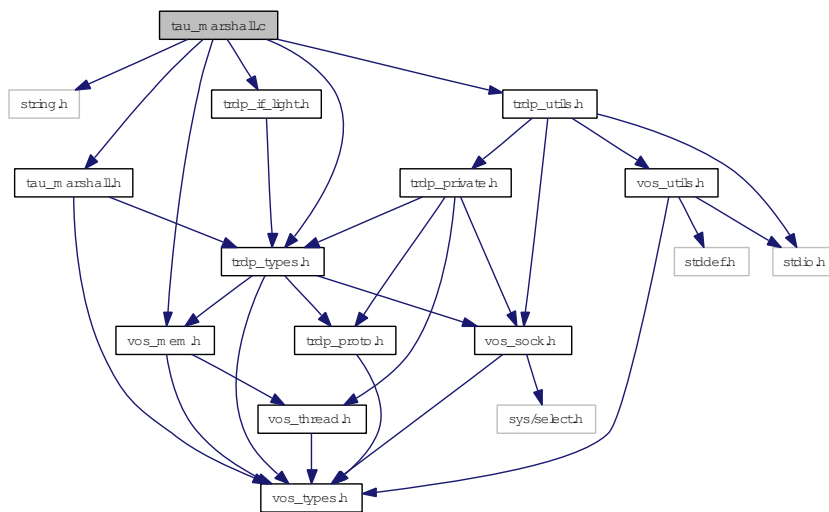
- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.2 tau_marshall.c File Reference

Marshalling functions for TRDP.

```
#include <string.h>
#include "trdp_types.h"
#include "trdp_if_light.h"
#include "trdp_utils.h"
#include "vos_mem.h"
#include "tau_marshall.h"
```

Include dependency graph for tau_marshall.c:



Data Structures

- struct [TAU_MARSHALL_INFO_T](#)
Marshalling info, used to and from wire.

Functions

- EXT_DECL [TRDP_ERR_T](#) [tau_initMarshall](#) (void **ppRefCon, UINT32 numComId, [TRDP_COMID_DSID_MAP_T](#) *pComIdDsIdMap, UINT32 numDataSet, [TRDP_DATASET_T](#) *pDataset[])

Function to initialise the marshalling/unmarshalling.
- EXT_DECL [TRDP_ERR_T](#) [tau_marshall](#) (void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize, [TRDP_DATASET_T](#) **ppDSPointer)

marshall function.
- EXT_DECL [TRDP_ERR_T](#) [tau_unmarshall](#) (void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize, [TRDP_DATASET_T](#) **ppDSPointer)

unmarshall function.

- EXT_DECL [TRDP_ERR_T tau_marshallDs](#) (void *pRefCon, UINT32 dsId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize, [TRDP_DATASET_T **ppDSPointer](#))

marshall data set function.

- EXT_DECL [TRDP_ERR_T tau_unmarshallDs](#) (void *pRefCon, UINT32 dsId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize, [TRDP_DATASET_T **ppDSPointer](#))

unmarshall data set function.

- EXT_DECL [TRDP_ERR_T tau_calcDatasetSize](#) (void *pRefCon, UINT32 dsId, UINT8 *pSrc, UINT32 *pDestSize, [TRDP_DATASET_T **ppDSPointer](#))

Calculate data set size by given data set id.

- EXT_DECL [TRDP_ERR_T tau_calcDatasetSizeByComId](#) (void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT32 *pDestSize, [TRDP_DATASET_T **ppDSPointer](#))

Calculate data set size by given ComId.

5.2.1 Detailed Description

Marshalling functions for TRDP.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[tau_marshall.c](#) 950 2013-06-13 13:51:41Z 97025

5.2.2 Function Documentation

5.2.2.1 EXT_DECL TRDP_ERR_T tau_calcDatasetSize (void *pRefCon, UINT32 dsId, UINT8 *pSrc, UINT32 *pDestSize, TRDP_DATASET_T **ppDSPointer)

Calculate data set size by given data set id.

Parameters:

- ← *pRefCon* Pointer to user context
- ← *dsId* Dataset id to identify the structure out of a configuration
- ← *pSrc* Pointer to received original message

- *pDestSize* Pointer to the size of the data set
- ↔ *ppDSPointer* pointer to pointer to cached dataset, set NULL if not used, set content NULL if unknown

Return values:

- TRDP_NO_ERR* no error
- TRDP_INIT_ERR* marshallng not initialised
- TRDP_PARAM_ERR* data set id not existing

5.2.2.2 EXT_DECL TRDP_ERR_T tau_calcDatasetSizeByComId (void * *pRefCon*, UINT32 *comId*, UINT8 * *pSrc*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

Calculate data set size by given ComId.

Parameters:

- ← *pRefCon* Pointer to user context
- ← *comId* ComId id to identify the structure out of a configuration
- ← *pSrc* Pointer to received original message
- *pDestSize* Pointer to the size of the data set
- ↔ *ppDSPointer* pointer to pointer to cached dataset, set NULL if not used, set content NULL if unknown

Return values:

- TRDP_NO_ERR* no error
- TRDP_INIT_ERR* marshallng not initialised
- TRDP_PARAM_ERR* data set id not existing

5.2.2.3 EXT_DECL TRDP_ERR_T tau_initMarshall (void ** *ppRefCon*, UINT32 *numComId*, TRDP_COMID_DSID_MAP_T * *pComIdDsIdMap*, UINT32 *numDataSet*, TRDP_DATASET_T * *pDataset*[])

Function to initialise the marshallng/unmarshallng.

Types for marshallng / unmarshallng.

The supplied array must be sorted by ComIds. The array must exist during the use of the marshallng functions (until [tlc_terminate\(\)](#)).

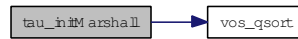
Parameters:

- ↔ *ppRefCon* Returns a pointer to be used for the reference context of marshallng/unmarshallng
- ← *numComId* Number of datasets found in the configuration
- ← *pComIdDsIdMap* Pointer to an array of structures of type TRDP_DATASET_T
- ← *numDataSet* Number of datasets found in the configuration
- ← *pDataset* Pointer to an array of pointers to 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

Here is the call graph for this function:



5.2.2.4 EXT_DECL TRDP_ERR_T tau_marshall (void * *pRefCon*, UINT32 *comId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

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
 ↔ *ppDSPointer* pointer to pointer to cached dataset set NULL if not used, set content NULL if unknown

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
TRDP_PARAM_ERR Parameter error

5.2.2.5 EXT_DECL TRDP_ERR_T tau_marshallDs (void * *pRefCon*, UINT32 *dsId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

marshall data set function.

Parameters:

← *pRefCon* pointer to user context
 ← *dsId* Data set id 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
 ↔ *ppDSPointer* pointer to pointer to cached dataset set NULL if not used, set content NULL if unknown

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
TRDP_PARAM_ERR Parameter error

5.2.2.6 EXT_DECL TRDP_ERR_T tau_unmarshall (void * *pRefCon*, UINT32 *comId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

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
 ↔ *ppDSPointer* pointer to pointer to cached dataset set NULL if not used, set content NULL if unknown

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

5.2.2.7 EXT_DECL TRDP_ERR_T tau_unmarshallDs (void * *pRefCon*, UINT32 *dsId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

unmarshall data set function.

Parameters:

← *pRefCon* pointer to user context
 ← *dsId* Data set id 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
 ↔ *ppDSPointer* pointer to pointer to cached dataset set NULL if not used, set content NULL if unknown

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

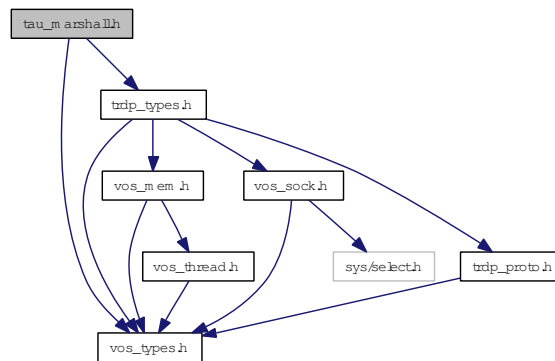
5.3 tau_marshall.h File Reference

TRDP utility interface definitions.

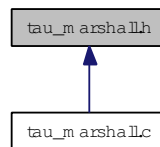
```
#include "vos_types.h"
```

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

Include dependency graph for tau_marshall.h:



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



Functions

- EXT_DECL [TRDP_ERR_T](#) [tau_initMarshall](#) (void **ppRefCon, UINT32 numComId, [TRDP_COMID_DSID_MAP_T](#) *pComIdDsIdMap, UINT32 numDataSet, [TRDP_DATASET_T](#) *pDataset[])

Types for marshalling / unmarshalling.
- EXT_DECL [TRDP_ERR_T](#) [tau_marshall](#) (void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize, [TRDP_DATASET_T](#) **ppDSPointer)

marshall function.
- EXT_DECL [TRDP_ERR_T](#) [tau_marshallDs](#) (void *pRefCon, UINT32 dsId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize, [TRDP_DATASET_T](#) **ppDSPointer)

marshall data set function.
- EXT_DECL [TRDP_ERR_T](#) [tau_unmarshall](#) (void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize, [TRDP_DATASET_T](#) **ppDSPointer)

unmarshall function.
- EXT_DECL [TRDP_ERR_T](#) [tau_unmarshallDs](#) (void *pRefCon, UINT32 dsId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize, [TRDP_DATASET_T](#) **ppDSPointer)

unmarshall data set function.

- EXT_DECL [TRDP_ERR_T tau_calcDatasetSize](#) (void *pRefCon, UINT32 dsId, UINT8 *pSrc, UINT32 *pDestSize, [TRDP_DATASET_T **ppDSPointer](#))

Calculate data set size by given data set id.

- EXT_DECL [TRDP_ERR_T tau_calcDatasetSizeByComId](#) (void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT32 *pDestSize, [TRDP_DATASET_T **ppDSPointer](#))

Calculate data set size by given ComId.

5.3.1 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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[tau_marshall.h](#) 676 2013-04-18 15:27:42Z bloehr

5.3.2 Function Documentation

5.3.2.1 EXT_DECL TRDP_ERR_T tau_calcDatasetSize (void *pRefCon, UINT32 dsId, UINT8 *pSrc, UINT32 *pDestSize, TRDP_DATASET_T **ppDSPointer)

Calculate data set size by given data set id.

Parameters:

- ← *pRefCon* Pointer to user context
- ← *dsId* Dataset id to identify the structure out of a configuration
- ← *pSrc* Pointer to received original message
- *pDestSize* Pointer to the size of the data set
- ↔ *ppDSPointer* pointer to pointer to cached dataset, set NULL if not used, set content NULL if unknown

Return values:

TRDP_NO_ERR no error
TRDP_INIT_ERR marshalling not initialised
TRDP_PARAM_ERR data set id not existing

5.3.2.2 EXT_DECL TRDP_ERR_T tau_calcDatasetSizeByComId (void * *pRefCon*, UINT32 *comId*, UINT8 * *pSrc*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

Calculate data set size by given ComId.

Parameters:

← *pRefCon* Pointer to user context
 ← *comId* ComId id to identify the structure out of a configuration
 ← *pSrc* Pointer to received original message
 → *pDestSize* Pointer to the size of the data set
 ↔ *ppDSPointer* pointer to pointer to cached dataset, set NULL if not used, set content NULL if unknown

Return values:

TRDP_NO_ERR no error
TRDP_INIT_ERR marshalling not initialised
TRDP_PARAM_ERR data set id not existing

5.3.2.3 EXT_DECL TRDP_ERR_T tau_initMarshall (void ** *ppRefCon*, UINT32 *numComId*, TRDP_COMID_DSID_MAP_T * *pComIdDsIdMap*, 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
 ← *numComId* Number of datasets found in the configuration
 ← *pComIdDsIdMap* Pointer to an array of structures of type TRDP_DATASET_T
 ← *numDataSet* Number of datasets found in the configuration
 ← *pDataset* Pointer to an array of pointers to 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

Types for marshalling / unmarshalling.

The supplied array must be sorted by ComIds. The array must exist during the use of the marshalling functions (until [tlc_terminate\(\)](#)).

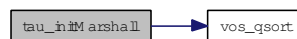
Parameters:

- ↔ *ppRefCon* Returns a pointer to be used for the reference context of marshalling/unmarshalling
- ← *numComId* Number of datasets found in the configuration
- ← *pComIdDsIdMap* Pointer to an array of structures of type TRDP_DATASET_T
- ← *numDataSet* Number of datasets found in the configuration
- ← *pDataset* Pointer to an array of pointers to 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

Here is the call graph for this function:



5.3.2.4 EXT_DECL TRDP_ERR_T tau_marshall (void * *pRefCon*, UINT32 *comId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

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
- ↔ *ppDSPointer* pointer to pointer to cached dataset set NULL if not used, set content NULL if unknown

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
- TRDP_PARAM_ERR* Parameter error

5.3.2.5 EXT_DECL TRDP_ERR_T tau_marshall (void * *pRefCon*, UINT32 *dsId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

marshall data set function.

Parameters:

- ← *pRefCon* pointer to user context
- ← *dsId* Data set id 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
- ↔ *ppDSPointer* pointer to pointer to cached dataset set NULL if not used, set content NULL if unknown

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
- TRDP_PARAM_ERR* Parameter error

5.3.2.6 EXT_DECL TRDP_ERR_T tau_unmarshall (void * *pRefCon*, UINT32 *comId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

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
- ↔ *ppDSPointer* pointer to pointer to cached dataset set NULL if not used, set content NULL if unknown

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

5.3.2.7 EXT_DECL TRDP_ERR_T tau_unmarshallDs (void * *pRefCon*, UINT32 *dsId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*, TRDP_DATASET_T ** *ppDSPointer*)

unmarshall data set function.

Parameters:

- ← *pRefCon* pointer to user context
- ← *dsId* Data set id 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
- ↔ *ppDSPointer* pointer to pointer to cached dataset set NULL if not used, set content NULL if unknown

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

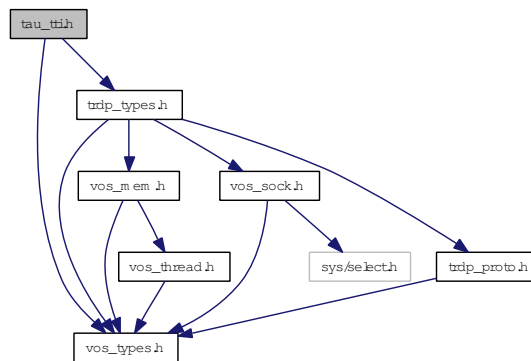
5.4 tau_tti.h File Reference

TRDP utility interface definitions.

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

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

Include dependency graph for tau_tti.h:



Data Structures

- struct [TRDP_FCT_INFO_T](#)
device information structure
- struct [TRDP_PROP_INFO_T](#)
properties information structure
- struct [TRDP_DEVICE_INFO_T](#)
device information structure
- struct [TRDP_CAR_INFO_T](#)
car information structure.
- struct [TRDP_CST_INFO_T](#)
consist information structure.
- struct [TRDP_TRAIN_INFO_T](#)
train information structure.

Enumerations

- enum [TRDP_INAUG_STATE_T](#) {
[TRDP_INAUG_INVALID](#),
[TRDP_INAUG_NOLEAD_UNCONF](#) = 2,
[TRDP_INAUG_LEAD_UNCONF](#) = 3,
[TRDP_INAUG_LEAD_CONF](#) = 4 }

Types for train configuration information.

- enum [TRDP_FCT_T](#) {
[TRDP_FCT_INVALID](#),
[TRDP_FCT_CAR](#) = 2,
[TRDP_FCT_CST](#) = 3,
[TRDP_FCT_TRAIN](#) = 4 }

function types

Functions

- EXT_DECL [TRDP_ERR_T](#) [tau_getEtbState](#) ([TRDP_INAUG_STATE_T](#) *pInaugState, UINT32 *pTopoCnt)

Function to retrieve the inauguration state and the topography counter.

- EXT_DECL [TRDP_ERR_T](#) [tau_getTrnCstCnt](#) (UINT16 *pTrnCstCnt, UINT32 *pTopoCnt)

Function to retrieve the total number of consists in the train.

- EXT_DECL [TRDP_ERR_T](#) [tau_getTrnCarCnt](#) (UINT16 *pTrnCarCnt, UINT32 *pTopoCnt)

Function to retrieve the total number of consists in the train.

- EXT_DECL [TRDP_ERR_T](#) [tau_getCstCarCnt](#) (UINT16 *pCstCarCnt, UINT32 *pTopoCnt, const [TRDP_LABEL_T](#) cstLabel)

Function to retrieve the total number of cars in a consist.

- EXT_DECL [TRDP_ERR_T](#) [tau_getCstFctCnt](#) (UINT16 *pCstFctCnt, UINT32 *pTopoCnt, const [TRDP_LABEL_T](#) cstLabel)

Function to retrieve the total number of functions in a consist.

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

Function to retrieve the total number of devices in a car.

- EXT_DECL [TRDP_ERR_T](#) [tau_getCstFctInfo](#) ([TRDP_FCT_INFO_T](#) *pFctInfo, UINT32 *pTopoCnt, const [TRDP_LABEL_T](#) cstLabel, UINT16 maxFctCnt)

Function to retrieve the function information of the consist.

- EXT_DECL [TRDP_ERR_T](#) [tau_getDevInfo](#) ([TRDP_DEV_INFO_T](#) *pDevInfo, UINT8 *pDevProp, UINT32 *pDevFctNo, UINT32 *pTopoCnt, const [TRDP_LABEL_T](#) devLabel, const [TRDP_LABEL_T](#) carLabel, const [TRDP_LABEL_T](#) cstLabel, UINT32 devPropLen, UINT16 devFctCnt)

Function to retrieve the device information of a car's device.

- EXT_DECL [TRDP_ERR_T](#) [tau_getCarInfo](#) ([TRDP_CAR_INFO_T](#) *pCarInfo, UINT8 *pCarProp, UINT32 *pTopoCnt, const [TRDP_LABEL_T](#) carLabel, const [TRDP_LABEL_T](#) cstLabel, UINT32 carPropLen)

Function to retrieve the car information of a consist's car.

- EXT_DECL `TRDP_ERR_T tau_getCstInfo (TRDP_CST_INFO_T *pCstInfo, UINT8 *pCstProp, UINT32 *pTopoCnt, const TRDP_LABEL_T cstLabel, UINT32 cstPropLen)`

Function to retrieve the consist information of a train's consist.

- EXT_DECL `TRDP_ERR_T tau_getTrnInfo (TRDP_CST_INFO_T *pTrnInfo, UINT32 *pTopoCnt)`

Function to retrieve the train information.

- *****
DECL `TRDP_ERR_T tau_getCarOrient (UINT8 *pCarOrient, UINT8 *pCstOrient, UINT32 *pTopoCnt, TRDP_LABEL_T carLabel, TRDP_LABEL_T cstLabel)`

Function to retrieve the orientation of the given car.

- EXT_DECL `TRDP_ERR_T tau_getIecCarOrient (UINT8 *pIecCarOrient, UINT8 *pIecCstOrient, UINT32 *pTopoCnt, TRDP_LABEL_T carLabel, TRDP_LABEL_T cstLabel)`

Function to retrieve the leading car depending IEC orientation of the given consist.

5.4.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- train topology information access

Note:

Project: TCNOpen TRDP prototype stack

Author:

Armin-H. Weiss (initial version)

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

tau_tci.h 274 2013-01-10 11:00:43Z aweiss

5.4.2 Enumeration Type Documentation

5.4.2.1 enum TRDP_FCT_T

function types

Enumerator:

TRDP_FCT_INVALID Invalid type.
Device local function

TRDP_FCT_CAR Car control function.
TRDP_FCT_CST Consist control function.
TRDP_FCT_TRAIN Train control function.

5.4.2.2 enum TRDP_INAUG_STATE_T

Types for train configuration information.

inauguration states

Enumerator:

TRDP_INAUG_INVALID Ongoing inauguration, DNS not yet available, no address transformation possible.
 Error in train inauguration, DNS not available, trainwide communication not possible
TRDP_INAUG_NOLEAD_UNCONF inauguration done, no leading vehicle set, inauguration unconfirmed
TRDP_INAUG_LEAD_UNCONF inauguration done, leading vehicle set, inauguration unconfirmed
TRDP_INAUG_LEAD_CONF inauguration done, leading vehicle set, inauguration confirmed

5.4.3 Function Documentation

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

Function to retrieve the total number of devices in a car.

Parameters:

→ *pDevCnt* Pointer to the device count to be returned
 ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
 ← *carLabel* Pointer to a car label. NULL means own car if *cstLabel* == NULL.
 ← *cstLabel* Pointer to a consist label. NULL means own consist.

Return values:

TRDP_NO_ERR no error
TRDP_PARAM_ERR Parameter error

5.4.3.2 EXT_DECL TRDP_ERR_T tau_getCarInfo (TRDP_CAR_INFO_T * *pCarInfo*, UINT8 * *pCarProp*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*, UINT32 *carPropLen*)

Function to retrieve the car information of a consist's car.

Parameters:

→ *pCarInfo* Pointer to the car info to be returned. Memory needs to be provided by application.

- *pCarProp* Pointer to application specific car properties to be returned. Memory needs to be provided by application. Set NULL if not used.
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* Pointer to a car label. NULL means own car if cstLabel refers to own consist.
- ← *cstLabel* Pointer to a consist label. NULL means own consist.
- ← *carPropLen* Length of provided buffer for car properties.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.4.3.3 *****

**EXT_DECL TRDP_ERR_T tau_getCarOrient (UINT8 * *pCarOrient*, UINT8 * *pCstOrient*,
UINT32 * *pTopoCnt*, TRDP_LABEL_T *carLabel*, TRDP_LABEL_T *cstLabel*)**

Function to retrieve the orientation of the given car.

Parameters:

- *pCarOrient* Pointer to the car orientation to be returned
- *pCstOrient* Pointer to the consist orientation to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* *carLabel* = NULL means own car if *cstLabel* == NULL
- ← *cstLabel* *cstLabel* = NULL means own consist

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.4.3.4 EXT_DECL TRDP_ERR_T tau_getCstCarCnt (UINT16 * *pCstCarCnt*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *cstLabel*)

Function to retrieve the total number of cars in a consist.

Parameters:

- *pCstCarCnt* Pointer to the number of cars to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *cstLabel* Pointer to a consist label. NULL means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.4.3.5 EXT_DECL TRDP_ERR_T tau_getCstFctCnt (UINT16 * *pCstFctCnt*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *cstLabel*)

Function to retrieve the total number of functions in a consist.

Parameters:

- *pCstFctCnt* Pointer to the number of functions to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *cstLabel* Pointer to a consist label. NULL means own consist.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.4.3.6 EXT_DECL TRDP_ERR_T tau_getCstFctInfo (TRDP_FCT_INFO_T * *pFctInfo*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *cstLabel*, UINT16 *maxFctCnt*)

Function to retrieve the function information of the consist.

Parameters:

- *pFctInfo* Pointer to function info list to be returned. Memory needs to be provided by application. Memory needs to be provided by application. Set NULL if not used.
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *cstLabel* Pointer to a consist label. NULL means own consist.
- ← *maxFctCnt* Maximal number of functions to be returned in provided buffer.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.4.3.7 EXT_DECL TRDP_ERR_T tau_getCstInfo (TRDP_CST_INFO_T * *pCstInfo*, UINT8 * *pCstProp*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *cstLabel*, UINT32 *cstPropLen*)

Function to retrieve the consist information of a train's consist.

Parameters:

- *pCstInfo* Pointer to the consist info to be returned. Memory needs to be provided by application.
- *pCstProp* Pointer to application specific consist properties to be returned. Memory needs to be provided by application. Set NULL if not used.
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *cstLabel* Pointer to a consist label. NULL means own consist.
- ← *cstPropLen* Length of provided buffer for consist properties.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.4.3.8 EXT_DECL TRDP_ERR_T tau_getDevInfo (TRDP_DEV_INFO_T * *pDevInfo*, UINT8 * *pDevProp*, UINT32 * *pDevFctNo*, UINT32 * *pTopoCnt*, const TRDP_LABEL_T *devLabel*, const TRDP_LABEL_T *carLabel*, const TRDP_LABEL_T *cstLabel*, UINT32 *devPropLen*, UINT16 *devFctCnt*)

Function to retrieve the device information of a car's device.

Parameters:

- *pDevInfo* Pointer to device infos to be returned. Memory needs to be provided by application.
- *pDevProp* Pointer to application specific device properties to be returned. Memory needs to be provided by application. Set NULL if not used.
- *pDevFctNo* Pointer to device function number list to be returned. Memory needs to be provided by application. Set NULL if not used.
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *devLabel* Pointer to a device label. NULL means own device if carLabel ist referring to own car. "devxxx" possible, with xxx = 001...999
- ← *carLabel* Pointer to a car label. NULL means own car if cstLabel refers to the own consist.
- ← *cstLabel* Pointer to a consist label. NULL means own consist.
- ← *devPropLen* Length of provided buffer for device properties.
- ← *devFctCnt* Maximal number of functions to be returned in provided buffer pDevFctNo.

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.4.3.9 EXT_DECL TRDP_ERR_T tau_getEtbState (TRDP_INAUG_STATE_T * *pInaugState*, UINT32 * *pTopoCnt*)

Function to retrieve the inauguration state and the topography counter.

Parameters:

- *pInaugState* Pointer to an inauguration state variable to be 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

5.4.3.10 EXT_DECL TRDP_ERR_T tau_getIecCarOrient (UINT8 * *pIecCarOrient*, UINT8 * *pIecCstOrient*, UINT32 * *pTopoCnt*, TRDP_LABEL_T *carLabel*, TRDP_LABEL_T *cstLabel*)

Function to retrieve the leading car depending IEC orientation of the given consist.

Parameters:

- *pIecCarOrient* Pointer to the IEC car orientation to be returned

- *pIecCstOrient* Pointer to the IEC consist orientation to be returned
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.
- ← *carLabel* carLabel = NULL means own car if cstLabel == NULL
- ← *cstLabel* cstLabel = NULL means own consist

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.4.3.11 EXT_DECL TRDP_ERR_T tau_getTrnCarCnt (UINT16 * *pTrnCarCnt*, UINT32 * *pTopoCnt*)

Function to retrieve the total number of consists in the train.

Parameters:

- *pTrnCarCnt* Pointer to the number of cars to be 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

5.4.3.12 EXT_DECL TRDP_ERR_T tau_getTrnCstCnt (UINT16 * *pTrnCstCnt*, UINT32 * *pTopoCnt*)

Function to retrieve the total number of consists in the train.

Parameters:

- *pTrnCstCnt* Pointer to the number of consists to be 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

5.4.3.13 EXT_DECL TRDP_ERR_T tau_getTrnInfo (TRDP_CST_INFO_T * *pTrnInfo*, UINT32 * *pTopoCnt*)

Function to retrieve the train information.

Parameters:

- *pTrnInfo* Pointer to the train info to be returned. Memory needs to be provided by application.
- ↔ *pTopoCnt* Pointer to the actual topo count. If !=0 will be checked. Returns the actual one.

Return values:

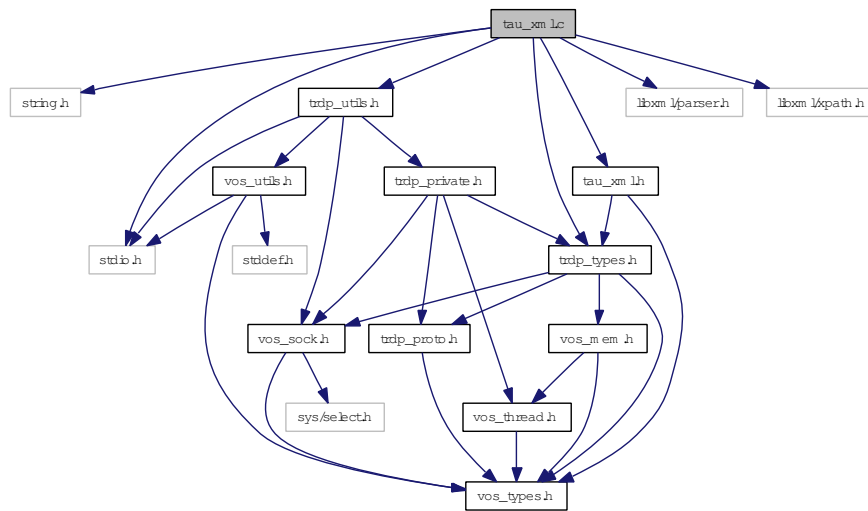
- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.5 tau_xml.c File Reference

Functions for XML file parsing.

```
#include <string.h>
#include <stdio.h>
#include "trdp_types.h"
#include "trdp_utils.h"
#include "tau_xml.h"
#include "libxml/parser.h"
#include "libxml/xpath.h"
```

Include dependency graph for tau_xml.c:



Defines

- `#define TRDP_SDT_DEFAULT_SMI2 0`
Default SDT safe message identifier.
- `#define TRDP_SDT_DEFAULT_NRXSAFE 3`
Default SDT timeout cycles.
- `#define TRDP_SDT_DEFAULT_NGUARD 100`
Default SDT initial timeout cycles.
- `#define TRDP_SDT_DEFAULT_CMTHR 10`
Default SDT chan.

Functions

- EXT_DECL [TRDP_ERR_T tau_prepareXmlDoc](#) (const [CHAR8 *pFileName](#), [TRDP_XML_DOC_HANDLE_T *pDocHnd](#))
Load XML file into DOM tree, prepare XPath context.
- EXT_DECL void [tau_freeXmlDoc](#) ([TRDP_XML_DOC_HANDLE_T *pDocHnd](#))
Free all the memory allocated by tau_prepareXmlDoc.
- EXT_DECL [TRDP_ERR_T tau_readXmlDeviceConfig](#) (const [TRDP_XML_DOC_HANDLE_T *pDocHnd](#), [TRDP_MEM_CONFIG_T *pMemConfig](#), [TRDP_DBG_CONFIG_T *pDbgConfig](#), [UINT32 *pNumComPar](#), [TRDP_COM_PAR_T **ppComPar](#), [UINT32 *pNumIfConfig](#), [TRDP_IF_CONFIG_T **ppIfConfig](#))
Function to read the TRDP device configuration parameters out of the XML configuration file.
- EXT_DECL [TRDP_ERR_T tau_readXmlDatasetConfig](#) (const [TRDP_XML_DOC_HANDLE_T *pDocHnd](#), [UINT32 *pNumComId](#), [TRDP_COMID_DSID_MAP_T **ppComIdDsIdMap](#), [UINT32 *pNumDataset](#), [papTRDP_DATASET_T papDataset](#))
Function to read the DataSet configuration out of the XML configuration file.
- EXT_DECL [TRDP_ERR_T tau_readXmlInterfaceConfig](#) (const [TRDP_XML_DOC_HANDLE_T *pDocHnd](#), const [CHAR8 *pIfName](#), [TRDP_PROCESS_CONFIG_T *pProcessConfig](#), [TRDP_PD_CONFIG_T *pPdConfig](#), [TRDP_MD_CONFIG_T *pMdConfig](#), [UINT32 *pNumExchgPar](#), [TRDP_EXCHG_PAR_T **ppExchgPar](#))
Read the interface relevant telegram parameters (except data set configuration) out of the configuration file
- EXT_DECL void [tau_freeTelegrams](#) ([UINT32 numExchgPar](#), [TRDP_EXCHG_PAR_T *pExchgPar](#))
Free array of telegram configurations allocated by tau_readXmlInterfaceConfig.

5.5.1 Detailed Description

Functions for XML file parsing.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Tomas Svoboda, UniContorls a.s.

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[tau_xml.c](#) 950 2013-06-13 13:51:41Z 97025

5.5.2 Define Documentation

5.5.2.1 #define TRDP_SDT_DEFAULT_CMTHR 10

Default SDT chan.
monitoring threshold

5.5.3 Function Documentation

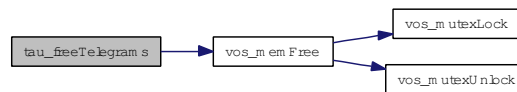
5.5.3.1 EXT_DECL void tau_freeTelegrams (UINT32 *numExchgPar*, TRDP_EXCHG_PAR_T * *pExchgPar*)

Free array of telegram configurations allocated by tau_readXmlInterfaceConfig.

Parameters:

- ← *numExchgPar* Number of telegram configurations in the array
- ← *pExchgPar* Pointer to array of telegram configurations

Here is the call graph for this function:



5.5.3.2 EXT_DECL void tau_freeXmlDoc (TRDP_XML_DOC_HANDLE_T * *pDocHnd*)

Free all the memory allocated by tau_prepareXmlDoc.

Parameters:

- ← *pDocHnd* Handle of the parsed XML file

5.5.3.3 EXT_DECL TRDP_ERR_T tau_prepareXmlDoc (const CHAR8 * *pFileName*, TRDP_XML_DOC_HANDLE_T * *pDocHnd*)

Load XML file into DOM tree, prepare XPath context.

Parameters:

- ← *pFileName* Path and filename of the xml configuration file
- *pDocHnd* Handle of the parsed XML file

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* File does not exist

5.5.3.4 EXT_DECL TRDP_ERR_T tau_readXmlDatasetConfig (const TRDP_XML_DOC_HANDLE_T * *pDocHnd*, UINT32 * *pNumComId*, TRDP_COMID_DSID_MAP_T ** *ppComIdDsIdMap*, UINT32 * *pNumDataset*, papTRDP_DATASET_T *papDataset*)

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

Parameters:

- ← *pDocHnd* Handle of the XML document prepared by tau_prepareXmlDoc
- *pNumComId* Pointer to the number of entries in the ComId DatasetId mapping list
- *ppComIdDsIdMap* Pointer to an array of a structures of type [TRDP_COMID_DSID_MAP_T](#)
- *pNumDataset* Pointer to the number of datasets found in the configuration
- *papDataset* Pointer to an array of pointers to 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

5.5.3.5 EXT_DECL TRDP_ERR_T tau_readXmlDeviceConfig (const TRDP_XML_DOC_HANDLE_T * *pDocHnd*, TRDP_MEM_CONFIG_T * *pMemConfig*, TRDP_DBG_CONFIG_T * *pDbgConfig*, UINT32 * *pNumComPar*, TRDP_COM_PAR_T ** *ppComPar*, UINT32 * *pNumIfConfig*, TRDP_IF_CONFIG_T ** *ppIfConfig*)

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

Parameters:

- ← *pDocHnd* Handle of the XML document prepared by tau_prepareXmlDoc
- *pMemConfig* Memory configuration
- *pDbgConfig* Debug printout configuration for application use
- *pNumComPar* Number of configured com parameters
- *ppComPar* Pointer to array of com parameters
- *pNumIfConfig* Number of configured interfaces
- *ppIfConfig* Pointer to an array of interface parameter sets

Return values:

- TRDP_NO_ERR* no error
- TRDP_MEM_ERR* provided buffer to small
- TRDP_PARAM_ERR* File not existing

5.5.3.6 EXT_DECL TRDP_ERR_T tau_readXmlInterfaceConfig (const TRDP_XML_DOC_HANDLE_T * *pDocHnd*, const CHAR8 * *pIfName*, TRDP_PROCESS_CONFIG_T * *pProcessConfig*, TRDP_PD_CONFIG_T * *pPdConfig*, TRDP_MD_CONFIG_T * *pMdConfig*, UINT32 * *pNumExchgPar*, TRDP_EXCHG_PAR_T ** *ppExchgPar*)

Read the interface relevant telegram parameters (except data set configuration) out of the configuration file

Parameters:

- ← *pDocHnd* Handle of the XML document prepared by tau_prepareXmlDoc
- ← *pIfName* Interface name
- *pProcessConfig* TRDP process (session) configuration for the interface
- *pPdConfig* PD default configuration for the interface
- *pMdConfig* MD default configuration for the interface
- *pNumExchgPar* Number of configured telegrams
- *ppExchgPar* Pointer to array of telegram configurations

Return values:

- TRDP_NO_ERR* no error
- TRDP_MEM_ERR* provided buffer too small
- TRDP_PARAM_ERR* File not existing

Here is the call graph for this function:



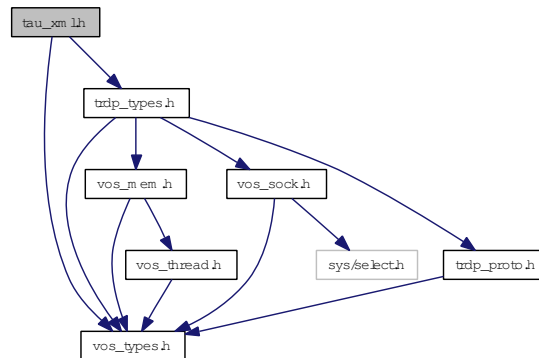
5.6 tau_xml.h File Reference

TRDP utility interface definitions.

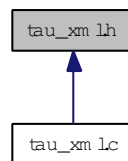
```
#include "vos_types.h"
```

```
#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_SDT_PAR_T](#)
Types to read out the XML configuration.
- struct [TRDP_DBG_CONFIG_T](#)
Control for debug output device/file on application level.
- struct [TRDP_XML_DOC_HANDLE_T](#)
Parsed XML document handle.

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_prepareXmlDoc](#) (const [CHAR8](#) *pFileName, [TRDP_XML_DOC_HANDLE_T](#) *pDocHnd)
Load XML file into DOM tree, prepare XPath context.
- EXT_DECL void [tau_freeXmlDoc](#) ([TRDP_XML_DOC_HANDLE_T](#) *pDocHnd)
Free all the memory allocated by tau_prepareXmlDoc.
- EXT_DECL [TRDP_ERR_T](#) [tau_readXmlDeviceConfig](#) (const [TRDP_XML_DOC_HANDLE_T](#) *pDocHnd, [TRDP_MEM_CONFIG_T](#) *pMemConfig, [TRDP_DBG_CONFIG_T](#) *pDbgConfig, [UINT32](#) *pNumComPar, [TRDP_COM_PAR_T](#) **ppComPar, [UINT32](#) *pNumIfConfig, [TRDP_IF_CONFIG_T](#) **ppIfConfig)
Function to read the TRDP device configuration parameters out of the XML configuration file.
- EXT_DECL [TRDP_ERR_T](#) [tau_readXmlInterfaceConfig](#) (const [TRDP_XML_DOC_HANDLE_T](#) *pDocHnd, const [CHAR8](#) *pIfName, [TRDP_PROCESS_CONFIG_T](#) *pProcessConfig, [TRDP_PD_CONFIG_T](#) *pPdConfig, [TRDP_MD_CONFIG_T](#) *pMdConfig, [UINT32](#) *pNumExchgPar, [TRDP_EXCHG_PAR_T](#) **ppExchgPar)
Read the interface relevant telegram parameters (except data set configuration) out of the configuration file.
- EXT_DECL [TRDP_ERR_T](#) [tau_readXmlDatasetConfig](#) (const [TRDP_XML_DOC_HANDLE_T](#) *pDocHnd, [UINT32](#) *pNumComId, [TRDP_COMID_DSID_MAP_T](#) **ppComIdDsIdMap, [UINT32](#) *pNumDataset, [papTRDP_DATASET_T](#) papDataset)
Function to read the DataSet configuration out of the XML configuration file.
- EXT_DECL void [tau_freeTelegrams](#) ([UINT32](#) numExchgPar, [TRDP_EXCHG_PAR_T](#) *pExchgPar)
Free array of telegram configurations allocated by tau_readXmlInterfaceConfig.

5.6.1 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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[tau_xml.h](#) 406 2013-01-25 16:28:16Z bloehr

5.6.2 Enumeration Type Documentation

5.6.2.1 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).

5.6.3 Function Documentation

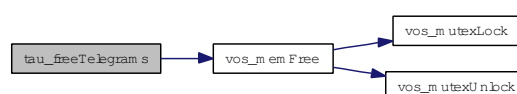
5.6.3.1 EXT_DECL void tau_freeTelegrams (UINT32 numExchgPar, TRDP_EXCHG_PAR_T * pExchgPar)

Free array of telegram configurations allocated by tau_readXmlInterfaceConfig.

Parameters:

← **numExchgPar** Number of telegram configurations in the array
 ← **pExchgPar** Pointer to array of telegram configurations

Here is the call graph for this function:



5.6.3.2 EXT_DECL void tau_freeXmlDoc (TRDP_XML_DOC_HANDLE_T * *pDocHnd*)

Free all the memory allocated by tau_prepareXmlDoc.

Parameters:

- ← *pDocHnd* Handle of the parsed XML file
- ← *pDocHnd* Handle of the parsed XML file

5.6.3.3 EXT_DECL TRDP_ERR_T tau_prepareXmlDoc (const CHAR8 * *pFileName*, TRDP_XML_DOC_HANDLE_T * *pDocHnd*)

Load XML file into DOM tree, prepare XPath context.

Parameters:

- ← *pFileName* Path and filename of the xml configuration file
- *pDocHnd* Handle of the parsed XML file

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* File does not exist

5.6.3.4 EXT_DECL TRDP_ERR_T tau_readXmlDatasetConfig (const TRDP_XML_DOC_HANDLE_T * *pDocHnd*, UINT32 * *pNumComId*, TRDP_COMID_DSID_MAP_T ** *ppComIdDsIdMap*, UINT32 * *pNumDataset*, papTRDP_DATASET_T *papDataset*)

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

Parameters:

- ← *pDocHnd* Handle of the XML document prepared by tau_prepareXmlDoc
- *pNumComId* Pointer to the number of entries in the ComId DatasetId mapping list
- *ppComIdDsIdMap* Pointer to an array of a structures of type [TRDP_COMID_DSID_MAP_T](#)
- *pNumDataset* Pointer to the number of datasets found in the configuration
- *papDataset* Pointer to an array of pointers to a structures of type TRDP_DATASET_T

Return values:

- TRDP_NO_ERR* no error
- TRDP_MEM_ERR* provided buffer too small
- TRDP_PARAM_ERR* File not existing

5.6.3.5 EXT_DECL TRDP_ERR_T tau_readXmlDeviceConfig (const TRDP_XML_DOC_HANDLE_T * *pDocHnd*, TRDP_MEM_CONFIG_T * *pMemConfig*, TRDP_DBG_CONFIG_T * *pDbgConfig*, UINT32 * *pNumComPar*, TRDP_COM_PAR_T ** *ppComPar*, UINT32 * *pNumIfConfig*, TRDP_IF_CONFIG_T ** *ppIfConfig*)

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

Parameters:

- ← *pDocHnd* Handle of the XML document prepared by tau_prepareXmlDoc
- *pMemConfig* Memory configuration
- *pDbgConfig* Debug printout configuration for application use
- *pNumComPar* Number of configured com parameters
- *ppComPar* Pointer to array of com parameters
- *pNumIfConfig* Number of configured interfaces
- *ppIfConfig* Pointer to an array of interface parameter sets

Return values:

- TRDP_NO_ERR* no error
- TRDP_MEM_ERR* provided buffer to small
- TRDP_PARAM_ERR* File not existing

5.6.3.6 EXT_DECL TRDP_ERR_T tau_readXmlInterfaceConfig (const TRDP_XML_DOC_HANDLE_T * *pDocHnd*, const CHAR8 * *pIfName*, TRDP_PROCESS_CONFIG_T * *pProcessConfig*, TRDP_PD_CONFIG_T * *pPdConfig*, TRDP_MD_CONFIG_T * *pMdConfig*, UINT32 * *pNumExchgPar*, TRDP_EXCHG_PAR_T ** *ppExchgPar*)

Read the interface relevant telegram parameters (except data set configuration) out of the configuration file .

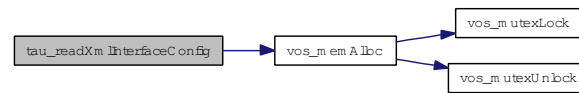
Parameters:

- ← *pDocHnd* Handle of the XML document prepared by tau_prepareXmlDoc
- ← *pIfName* Interface name
- *pProcessConfig* TRDP process (session) configuration for the interface
- *pPdConfig* PD default configuration for the interface
- *pMdConfig* MD default configuration for the interface
- *pNumExchgPar* Number of configured telegrams
- *ppExchgPar* Pointer to array of telegram configurations

Return values:

- TRDP_NO_ERR* no error
- TRDP_MEM_ERR* provided buffer to small
- TRDP_PARAM_ERR* File not existing

Here is the call graph for this function:



5.7 trdp_dllmain.c File Reference

Windows DLL main function.

5.7.1 Detailed Description

Windows DLL main function.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Armin-H. Weiss, Bombardier

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

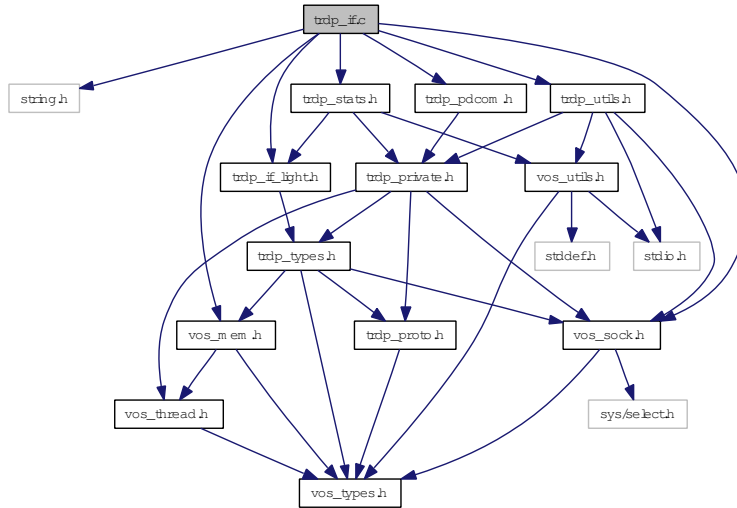
[trdp_dllmain.c](#) 950 2013-06-13 13:51:41Z 97025

5.8 trdp_if.c File Reference

Functions for ECN communication.

```
#include <string.h>
#include "trdp_if_light.h"
#include "trdp_utils.h"
#include "trdp_pdcom.h"
#include "trdp_stats.h"
#include "vos_sock.h"
#include "vos_mem.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](#) (const [TRDP_PRINT_DBG_T](#) pPrintDebugString, const [TRDP_MEM_CONFIG_T](#) *pMemConfig)
Initialize the TRDP stack.
- **EXT_DECL** [TRDP_ERR_T](#) [tlc_openSession](#) ([TRDP_APP_SESSION_T](#) *pAppHandle, [TRDP_IP_ADDR_T](#) ownIpAddr, [TRDP_IP_ADDR_T](#) leaderIpAddr, const [TRDP_MARSHALL_CONFIG_T](#) *pMarshall, const [TRDP_PD_CONFIG_T](#) *pPdDefault, const [TRDP_MD_CONFIG_T](#) *pMdDefault, const [TRDP_PROCESS_CONFIG_T](#) *pProcessConfig)
Open a session with the TRDP stack.

- EXT_DECL [TRDP_ERR_T tlc_closeSession](#) ([TRDP_APP_SESSION_T](#) appHandle)
Close a session.
- EXT_DECL [TRDP_ERR_T tlc_terminate](#) (void)
Un-Initialize.
- EXT_DECL [TRDP_ERR_T tlc_reinitSession](#) ([TRDP_APP_SESSION_T](#) appHandle)
Re-Initialize.
- const char * [tlc_getVersionString](#) (void)
Return a human readable version representation.
- EXT_DECL const [TRDP_VERSION_T](#) * [tlc_getVersion](#) (void)
Return version.
- [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 tlc_setTopoCount](#) ([TRDP_APP_SESSION_T](#) appHandle, 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)
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_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)

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, [TRDP_FLAGS_T](#) pktFlags, [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_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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_if.c](#) 950 2013-06-13 13:51:41Z 97025

BL 2013-02-01: ID 53: Zero dataset size fixed for PD

BL 2013-01-25: ID 20: Redundancy handling fixed

BL 2013-01-08: LADDER: Removed/Changed some ladder specific code in [tlp_subscribe\(\)](#)

BL 2012-12-03: ID 1: "using uninitialized PD_ELE_T.pullIpAddress variable" ID 2: "uninitialized PD_ELE_T newPD → pNext in tlp_subscribe()"

5.8.2 Function Documentation

5.8.2.1 EXT_DECL TRDP_ERR_T tlc_closeSession (TRDP_APP_SESSION_T appHandle)

Close a session.

Clean up and release all resources of that session

Parameters:

← *appHandle* The handle returned by tlc_openSession

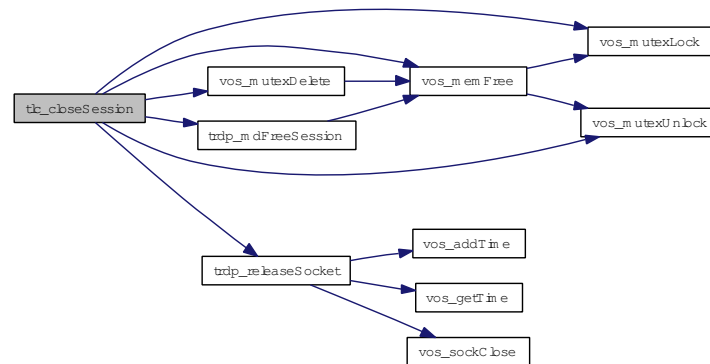
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Here is the call graph for this function:



5.8.2.2 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_openSession

→ *pInterval* pointer to needed interval

↔ *pFileDesc* pointer to file descriptor set

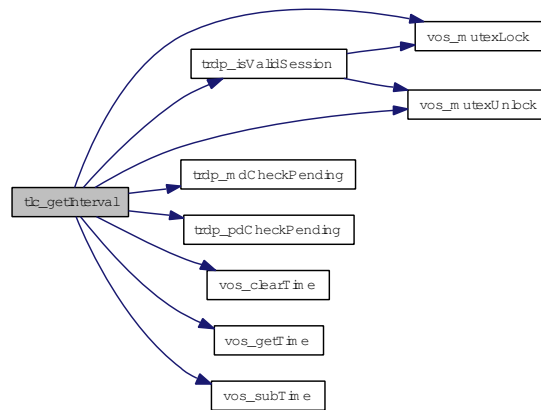
→ *pNoDesc* pointer to put no of highest used descriptors (for select())

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.3 EXT_DECL const TRDP_VERSION_T* tlc_getVersion (void)

Return version.

Return pointer to version structure

Return values:

TRDP_VERSION_T

5.8.2.4 const char* tlc_getVersionString (void)

Return a human readable version representation.

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

Return values:

const string

5.8.2.5 EXT_DECL TRDP_ERR_T tlc_init (const TRDP_PRINT_DBG_T pPrintDebugString, const TRDP_MEM_CONFIG_T * pMemConfig)

Initialize the TRDP stack.

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

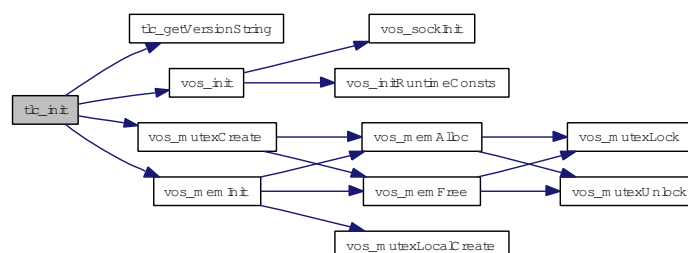
Parameters:

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

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

Return values:*TRDP_NO_ERR* no error*TRDP_MEM_ERR* memory allocation failed*TRDP_PARAM_ERR* initialization error

Here is the call graph for this function:



5.8.2.6 `EXT_DECL TRDP_ERR_T tlc_openSession (TRDP_APP_SESSION_T * pAppHandle, TRDP_IP_ADDR_T ownIpAddr, TRDP_IP_ADDR_T leaderIpAddr, const TRDP_MARSHALL_CONFIG_T * pMarshall, const TRDP_PD_CONFIG_T * pPdDefault, const TRDP_MD_CONFIG_T * pMdDefault, const TRDP_PROCESS_CONFIG_T * pProcessConfig)`

Open a session with the TRDP stack.

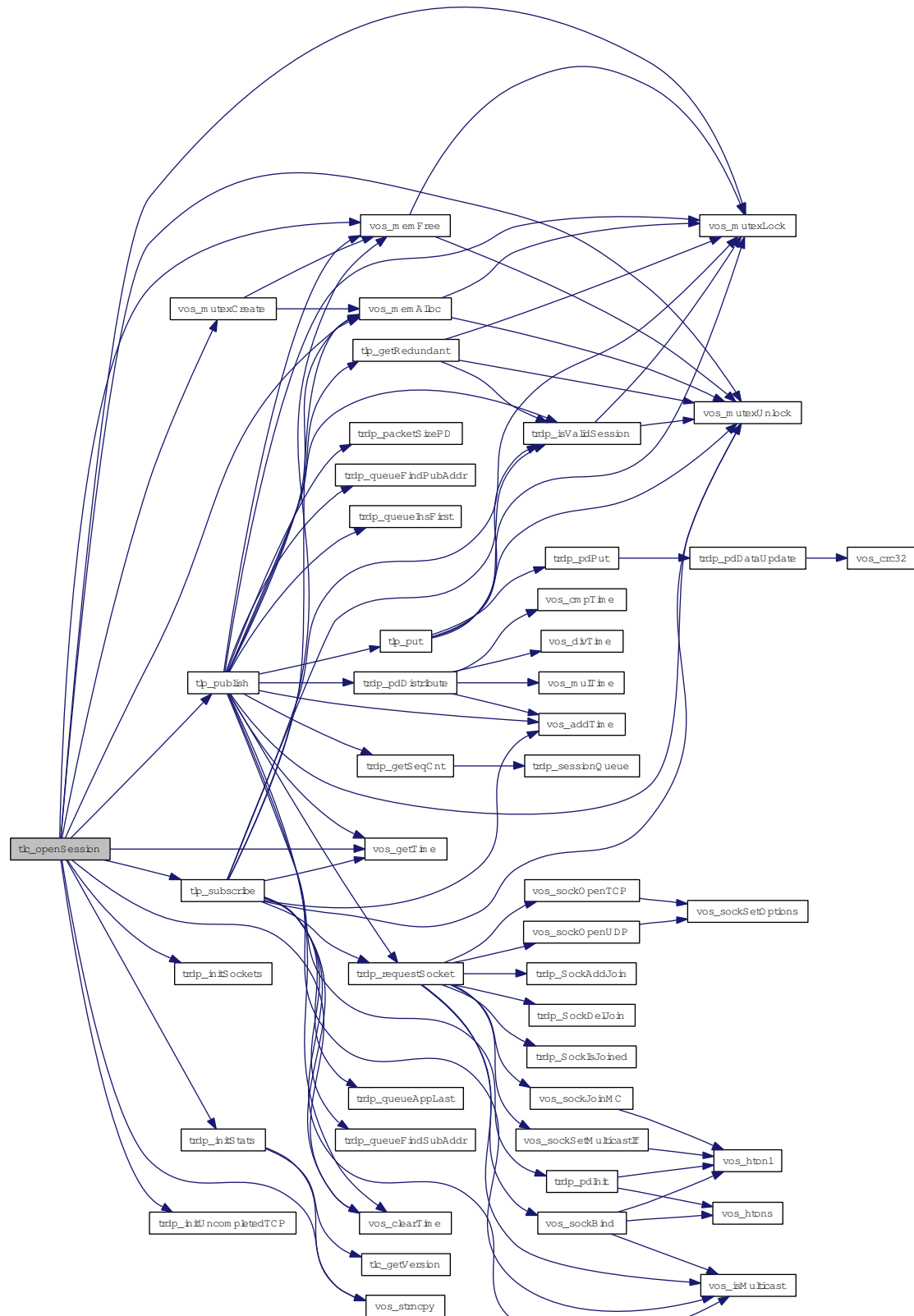
`tlc_openSession` returns in `pAppHandle` a unique handle to be used in further calls to the stack.

Parameters:

- *pAppHandle* A handle for further calls to the trdp stack
- ← *ownIpAddr* Own IP address, can be different for each process in multihoming systems, if zero, the default interface / IP will be used.
- ← *leaderIpAddr* IP address of redundancy leader
- ← *pMarshall* Pointer to marshalling configuration
- ← *pPdDefault* Pointer to default PD configuration
- ← *pMdDefault* Pointer to default MD configuration
- ← *pProcessConfig* Pointer to process configuration only option parameter is used here to define session behavior all other parameters are only used to feed statistics

Return values:*TRDP_NO_ERR* no error*TRDP_INIT_ERR* not yet initied*TRDP_PARAM_ERR* parameter error*TRDP SOCK_ERR* socket error

Here is the call graph for this function:



5.8.2.7 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_openSession

← *pRfds* pointer to set of ready descriptors

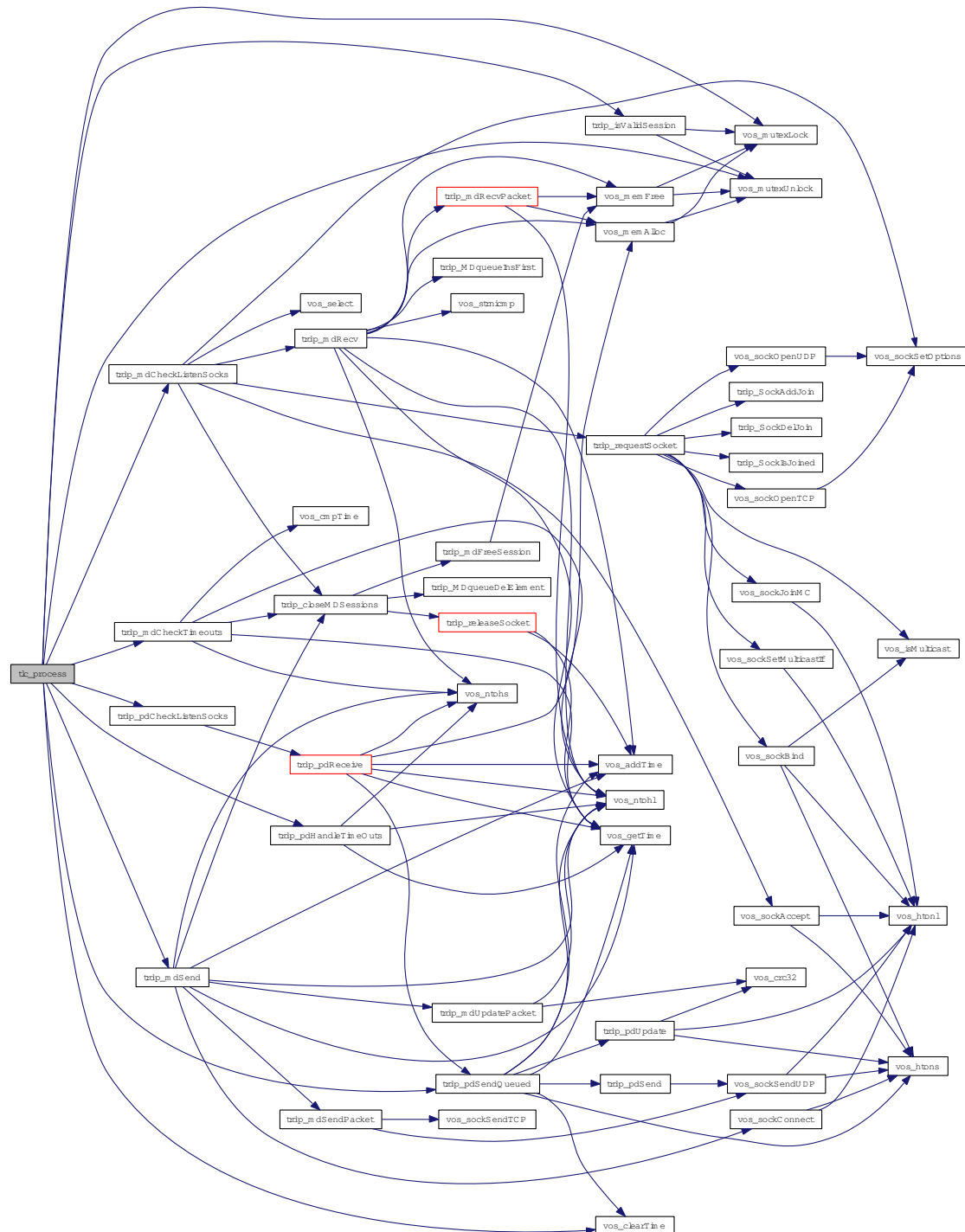
↔ *pCount* pointer to number of ready descriptors

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.8 EXT_DECL TRDP_ERR_T tlc_reinitSession (TRDP_APP_SESSION_T *appHandle*)

Re-Initialize.

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

Parameters:

← *appHandle* The handle returned by tlc_openSession

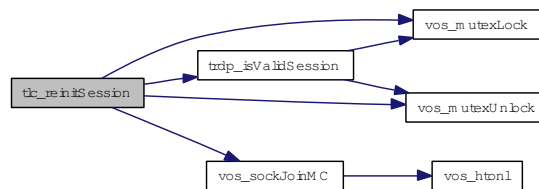
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Here is the call graph for this function:



5.8.2.9 EXT_DECL TRDP_ERR_T tlc_setTopoCount (TRDP_APP_SESSION_T *appHandle*, UINT32 *topoCount*)

Set new topoCount for trainwide communication.

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

Parameters:

← *appHandle* the handle returned by tlc_openSession

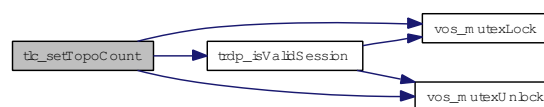
← *topoCount* New topoCount value

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.10 EXT_DECL TRDP_ERR_T tlc_terminate (void)

Un-Initialize.

Clean up and close all sessions. Mainly used for debugging/test runs. No further calls to library allowed

Return values:

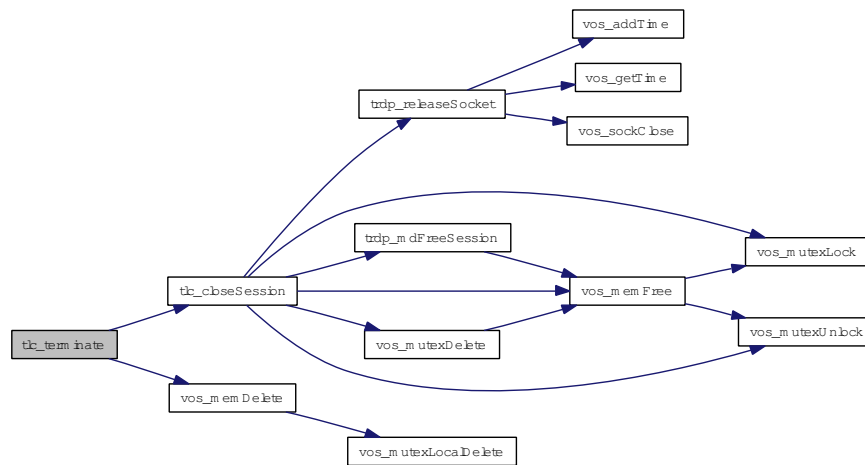
TRDP_NO_ERR no error

TRDP_INIT_ERR no error

TRDP_MEM_ERR TrafficStore nothing

TRDP_MUTEX_ERR TrafficStore mutex err

Here is the call graph for this function:



5.8.2.11 EXT_DECL TRDP_ERR_T tlp_get (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T subHandle, 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_openSession`

← **subHandle** the handle returned by subscription

↔ **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

Only the status of the first redundancy group entry is returned will be returned!

Parameters:

- ← **appHandle** the handle returned by tlc_init
- ← **redId** will be returned for all ComID's with the given redId
- ↔ **pLeader** TRUE if we're sending this redundancy group (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:



5.8.2.13 `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)`

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_openSession
- **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, 0 if PD PULL
- ← **redId** 0 - Non-redundant, > 0 valid redundancy group
- ← **pktFlags** OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_NONE, 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

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

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

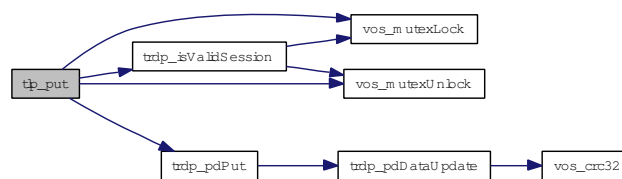
Parameters:

- ← **appHandle** the handle returned by tlc_openSession
- ← **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 on uninitialized parameter or changed dataSize compared to published one
- TRDP_NOPUB_ERR** not published
- TRDP_NOINIT_ERR** handle invalid
- TRDP_COMID_ERR** ComID not found when marshalling

Here is the call graph for this function:



5.8.2.15 `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)`

Initiate sending PD messages (PULL).

Send a PD request message

Parameters:

- ← **appHandle** the handle returned by tlc_openSession
- ← **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** OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_NONE, TRDP_FLAGS_-MARSHALL, TRDP_FLAGS_CALLBACK

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

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

Do not send redundant PD's 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:



5.8.2.17 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*, TRDP_FLAGS_T *pktFlags*, UINT32 *timeout*, TRDP_TO_BEHAVIOR_T *toBehavior*, UINT32 *maxDataSize*)

Prepare for receiving PD messages.

Subscribe to a specific PD ComID and source IP.

Parameters:

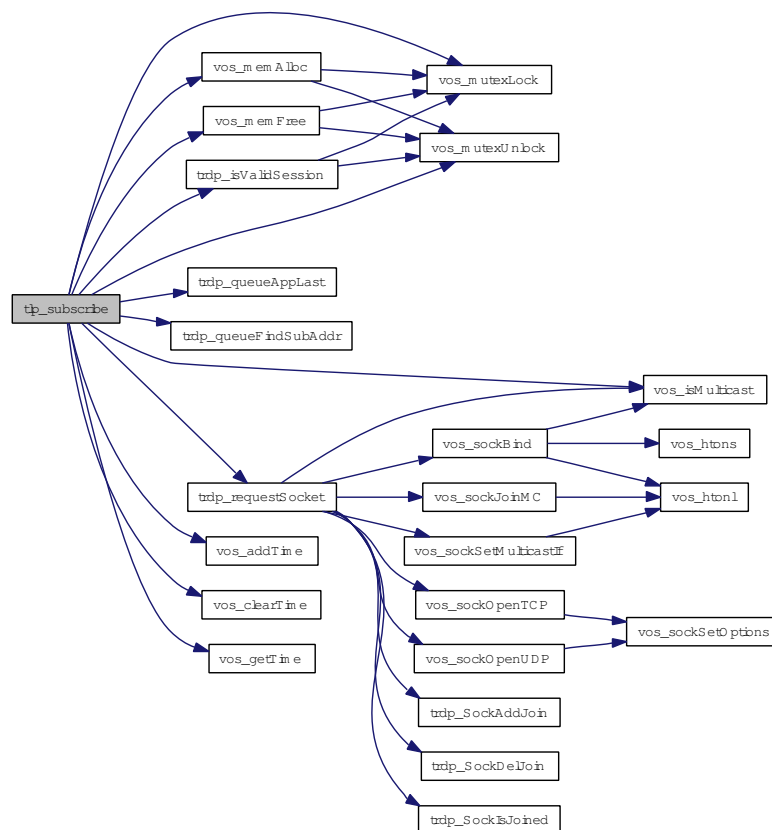
- ← *appHandle* the handle returned by tlc_openSession
- *pSubHandle* return a handle for these messages
- ← *pUserRef* user supplied value returned within the info structure
- ← *comId* comId of packet to receive
- ← *topoCount* valid topocount, 0 for local consist
- ← *srcIpAddr1* IP for source filtering, 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.
- ← *pktFlags* OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_NONE, TRDP_FLAGS_-MARSHALL, TRDP_FLAGS_CALLBACK
- ← *destIpAddr* IP address to join

- ← *timeout* timeout (≥ 10 ms) 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:



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

Stop sending PD messages.

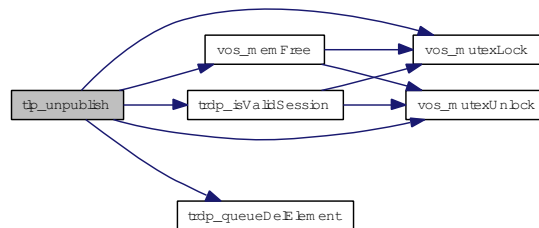
Parameters:

- ← *appHandle* the handle returned by tlc_openSession
- ← *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:



5.8.2.19 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

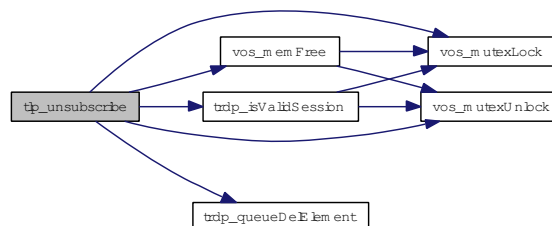
Parameters:

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

Return values:

TRDP_NO_ERR no error
TRDP_PARAM_ERR parameter error
TRDP_NOSUB_ERR not subscribed
TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.8.2.20 BOOL trdp_isValidSession (TRDP_APP_SESSION_T *pSessionHandle*)

Check if the session handle is valid.

Parameters:

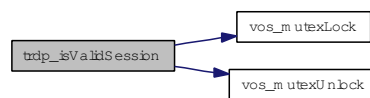
← *pSessionHandle* pointer to packet data (dataset)

Return values:

TRUE is valid

FALSE is invalid

Here is the call graph for this function:



5.8.2.21 TRDP_APP_SESSION_T* trdp_sessionQueue (void)

Get the session queue head pointer.

Return values:

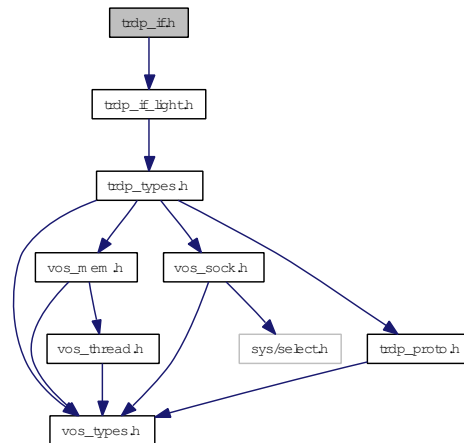
&sSession

5.9 trdp_if.h File Reference

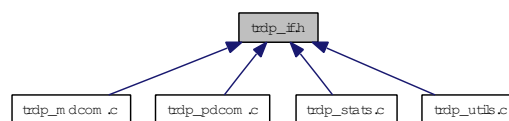
Typedefs for TRDP communication.

```
#include "trdp_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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_if.h](#) 950 2013-06-13 13:51:41Z 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

Parameters:

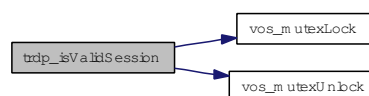
← *pSessionHandle* pointer to packet data (dataset)

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

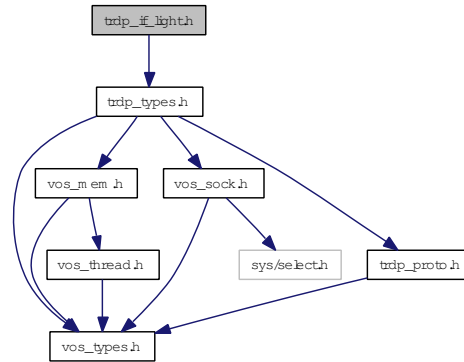
&sSession

5.10 trdp_if_light.h File Reference

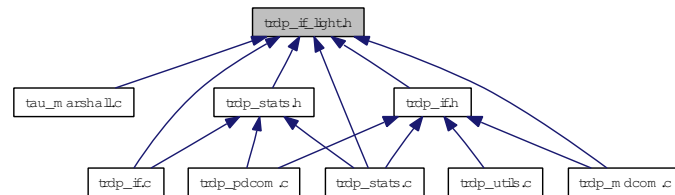
TRDP Light interface functions (API).

```
#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 (const TRDP_PRINT_DBG_T pPrintDebugString, const TRDP_MEM_CONFIG_T *pMemConfig)`
Initialize the TRDP stack.
- `EXT_DECL TRDP_ERR_T tlc_openSession (TRDP_APP_SESSION_T *pAppHandle, TRDP_IP_ADDR_T ownIpAddr, TRDP_IP_ADDR_T leaderIpAddr, const TRDP_MARSHALL_CONFIG_T *pMarshall, const TRDP_PD_CONFIG_T *pPdDefault, const TRDP_MD_CONFIG_T *pMdDefault, const TRDP_PROCESS_CONFIG_T *pProcessConfig)`
Open a session with the TRDP stack.
- `EXT_DECL TRDP_ERR_T tlc_reinitSession (TRDP_APP_SESSION_T appHandle)`
Re-Initialize.

- EXT_DECL [TRDP_ERR_T tlc_closeSession](#) ([TRDP_APP_SESSION_T](#) appHandle)
Close a session.
- EXT_DECL [TRDP_ERR_T tlc_terminate](#) (void)
Un-Initialize.
- EXT_DECL [TRDP_ERR_T tlc_setTopoCount](#) ([TRDP_APP_SESSION_T](#) appHandle, [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)
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 redundant PD's 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)
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, [TRDP_FLAGS_T](#) pktFlags, 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_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 numReplies, UINT32 replyTimeout, 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 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](#) sourceURI, const [TRDP_URI_USER_T](#) destURI)

Initiate sending MD confirm message.

- EXT_DECL [TRDP_ERR_T](#) [tlm_abortSession](#) ([TRDP_APP_SESSION_T](#) appHandle, const [TRDP_UUID_T](#) *pSessionId)

Cancel an open session.

- EXT_DECL [TRDP_ERR_T](#) [tlm_addListener](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_LIS_T](#) *pListenHandle, const void *pUserRef, UINT32 comId, UINT32 topoCount, [TRDP_IP_ADDR_T](#) mcDestIpAddr, [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, [TRDP_LIS_T](#) listenHandle)

Remove Listener.

- EXT_DECL [TRDP_ERR_T](#) [tlm_reply](#) ([TRDP_APP_SESSION_T](#) appHandle, void *pUserRef, const [TRDP_UUID_T](#) *pSessionId, UINT32 topoCount, UINT32 comId, [TRDP_IP_ADDR_T](#) srcIpAddr, [TRDP_IP_ADDR_T](#) destIpAddr, [TRDP_FLAGS_T](#) pktFlags, UINT16 userStatus, const

[TRDP_SEND_PARAM_T](#) *pSendParam, const UINT8 *pData, UINT32 dataSize, const [TRDP_URI_USER_T](#) sourceURI, const [TRDP_URI_USER_T](#) destURI)

Send a MD reply message.

- EXT_DECL [TRDP_ERR_T](#) [tlm_replyQuery](#) ([TRDP_APP_SESSION_T](#) appHandle, void *pUserRef, const [TRDP_UUID_T](#) *pSessionId, UINT32 topoCount, UINT32 comId, [TRDP_IP_ADDR_T](#) srcIpAddr, [TRDP_IP_ADDR_T](#) destIpAddr, [TRDP_FLAGS_T](#) pktFlags, UINT16 userStatus, UINT32 confirmTimeout, const [TRDP_SEND_PARAM_T](#) *pSendParam, const UINT8 *pData, UINT32 dataSize, const [TRDP_URI_USER_T](#) sourceURI, const [TRDP_URI_USER_T](#) destURI)

Send a MD reply message.

- EXT_DECL [TRDP_ERR_T](#) [tlm_replyErr](#) ([TRDP_APP_SESSION_T](#) appHandle, const [TRDP_UUID_T](#) *pSessionId, UINT32 topoCount, UINT32 comId, [TRDP_IP_ADDR_T](#) srcIpAddr, [TRDP_IP_ADDR_T](#) destIpAddr, [TRDP_REPLY_STATUS_T](#) replyState, const [TRDP_SEND_PARAM_T](#) *pSendParam, const [TRDP_URI_USER_T](#) sourceURI, const [TRDP_URI_USER_T](#) destURI)

Send a MD error reply message.

- EXT_DECL const CHAR8 * [tlc_getVersionString](#) (void)

Return a human readable version representation.

- EXT_DECL const [TRDP_VERSION_T](#) * [tlc_getVersion](#) (void)

Return version.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_STATISTICS_T](#) *pStatistics)

Return statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getSubsStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumSubs, [TRDP_SUBS_STATISTICS_T](#) *pStatistics)

Return PD subscription statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getPubStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumPub, [TRDP_PUB_STATISTICS_T](#) *pStatistics)

Return PD publish statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getListStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumList, [TRDP_LIST_STATISTICS_T](#) *pStatistics)

Return MD listener statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getRedStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumRed, [TRDP_RED_STATISTICS_T](#) *pStatistics)

Return redundancy group statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getJoinStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumJoin, UINT32 *pIpAddr)

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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_if_light.h](#) 897 2013-06-05 15:03:51Z bloehr

5.10.2 Function Documentation

5.10.2.1 EXT_DECL TRDP_ERR_T tlc_closeSession (TRDP_APP_SESSION_T *appHandle*)

Close a session.

Clean up and release all resources of that session

Parameters:

← *appHandle* The handle returned by tlc_openSession

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Clean up and release all resources of that session

Parameters:

← *appHandle* The handle returned by tlc_openSession

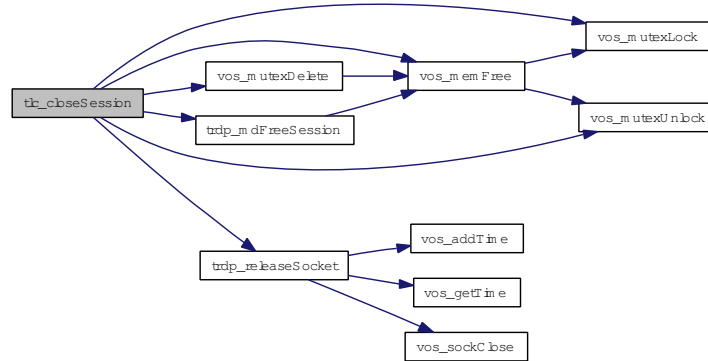
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Here is the call graph for this function:



5.10.2.2 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

5.10.2.3 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

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

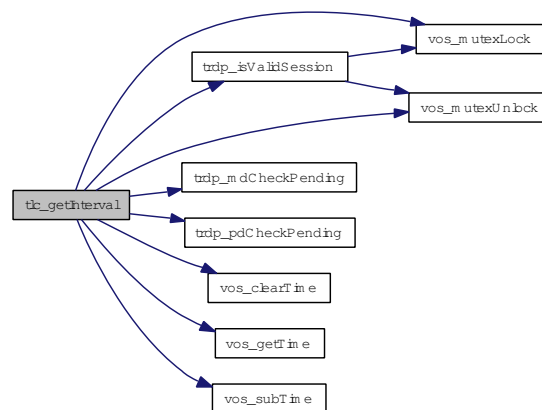
Parameters:

- ← **appHandle** The handle returned by tlc_openSession
- **pInterval** pointer to needed interval
- ↔ **pFileDesc** pointer to file descriptor set
- **pNoDesc** pointer to put no of highest used descriptors (for select())

Return values:

- TRDP_NO_ERR** no error
- TRDP_NOINIT_ERR** handle invalid

Here is the call graph for this function:



5.10.2.4 EXT_DECL TRDP_ERR_T tlc_getJoinStatistics (TRDP_APP_SESSION_T appHandle, UINT16 *pNumJoin, UINT32 *pIpAddr)

Return join statistics.

Memory for statistics information must be provided by the user. must be provided by the user. The reserved length is given via pNumJoin implicitly.

Parameters:

- ← **appHandle** the handle returned by tlc_openSession
- ↔ **pNumJoin** Pointer to the number of joined IP Adresses
- **pIpAddr** Pointer to a list with the joined IP addresses

Return values:

- TRDP_NO_ERR** no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR there are more items than requested

Memory for statistics information must be provided by the user.

Parameters:

← **appHandle** the handle returned by `tlc_openSession`

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

→ **pIpAddr** Pointer to a list with the joined IP addresses

Return values:

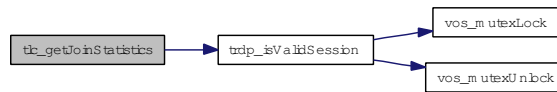
TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR there are more items than requested

Here is the call graph for this function:



5.10.2.5 EXT_DECL TRDP_ERR_T tlc_getListStatistics (TRDP_APP_SESSION_T appHandle, UINT16 * pNumList, TRDP_LIST_STATISTICS_T * pStatistics)

Return MD listener statistics.

Memory for statistics information must be provided by the user. The reserved length is given via `pNumList` implicitly.

Parameters:

← **appHandle** the handle returned by `tlc_openSession`

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

→ **pStatistics** 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

TRDP_MEM_ERR there are more subscriptions than requested

Memory for statistics information must be provided by the user.

Parameters:

- ← *appHandle* the handle returned by `tlc_openSession`
- ↔ *pNumList* Pointer to the number of listeners
- *pStatistics* 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
- TRDP_MEM_ERR** there are more subscriptions than requested

Here is the call graph for this function:



5.10.2.6 EXT_DECL TRDP_ERR_T tlc_getPubStatistics (TRDP_APP_SESSION_T appHandle, UINT16 * pNumPub, TRDP_PUB_STATISTICS_T * pStatistics)

Return PD publish statistics.

Memory for statistics information must be provided by the user. The reserved length is given via `pNumPub` implicitly.

Parameters:

- ← *appHandle* the handle returned by `tlc_openSession`
- ↔ *pNumPub* Pointer to the number of publishers
- *pStatistics* 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
- TRDP_MEM_ERR** there are more subscriptions than requested

Memory for statistics information must be provided by the user.

Parameters:

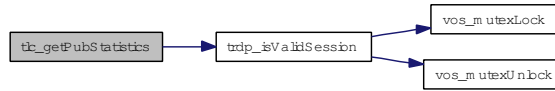
- ← *appHandle* the handle returned by `tlc_openSession`
- ↔ *pNumPub* Pointer to the number of publishers
- *pStatistics* Pointer to a list with the publish statistics information

Return values:

- TRDP_NO_ERR** no error

TRDP_NOINIT_ERR handle invalid
TRDP_PARAM_ERR parameter error
TRDP_MEM_ERR there are more subscriptions than requested

Here is the call graph for this function:



5.10.2.7 EXT_DECL TRDP_ERR_T tlc_getRedStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 **pNumRed*, TRDP_RED_STATISTICS_T **pStatistics*)

Return redundancy group statistics.

Memory for statistics information must be provided by the user. The reserved length is given via *pNumRed* implicitly.

Parameters:

← ***appHandle*** the handle returned by `tlc_openSession`
 ↔ ***pNumRed*** Pointer to the number of redundancy groups
 → ***pStatistics*** 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
TRDP_MEM_ERR there are more subscriptions than requested

Memory for statistics information must be provided by the user.

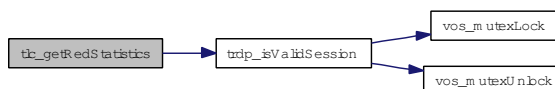
Parameters:

← ***appHandle*** the handle returned by `tlc_openSession`
 ↔ ***pNumRed*** Pointer to the number of redundancy groups
 → ***pStatistics*** 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
TRDP_MEM_ERR there are more subscriptions than requested

Here is the call graph for this function:



5.10.2.8 EXT_DECL TRDP_ERR_T tlc_getStatistics (TRDP_APP_SESSION_T *appHandle*, TRDP_STATISTICS_T **pStatistics*)

Return statistics.

Memory for statistics information must be preserved by the user.

Parameters:

- ← *appHandle* the handle returned by tlc_init
- *pStatistics* Pointer to statistics for this application session

Return values:

- TRDP_NO_ERR** no error
- TRDP_NOINIT_ERR** handle invalid
- TRDP_PARAM_ERR** parameter error

Memory for statistics information must be provided by the user.

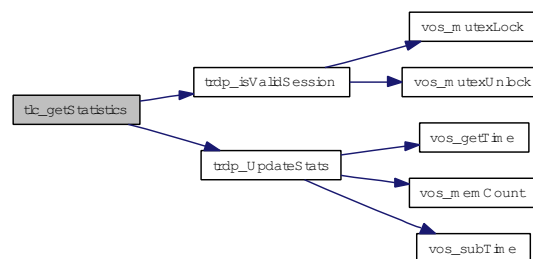
Parameters:

- ← *appHandle* the handle returned by tlc_openSession
- *pStatistics* Pointer to 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:



5.10.2.9 EXT_DECL TRDP_ERR_T tlc_getSubsStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 **pNumSubs*, TRDP_SUBS_STATISTICS_T **pStatistics*)

Return PD subscription statistics.

Memory for statistics information must be provided by the user. The reserved length is given via `pNumSub` implicitly.

Parameters:

- ← *appHandle* the handle returned by tlc_openSession

↔ *pNumSubs* In: The number of subscriptions requested Out: Number of subscriptions returned
 ↔ *pStatistics* Pointer to an array with the subscription statistics information

Return values:

TRDP_NO_ERR no error
TRDP_NOINIT_ERR handle invalid
TRDP_PARAM_ERR parameter error
TRDP_MEM_ERR there are more subscriptions than requested

Memory for statistics information must be provided by the user.

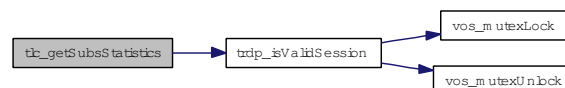
Parameters:

← *appHandle* the handle returned by tlc_openSession
 ↔ *pNumSubs* In: The number of subscriptions requested Out: Number of subscriptions returned
 ↔ *pStatistics* Pointer to an array with the subscription statistics information

Return values:

TRDP_NO_ERR no error
TRDP_NOINIT_ERR handle invalid
TRDP_PARAM_ERR parameter error
TRDP_MEM_ERR there are more subscriptions than requested

Here is the call graph for this function:



5.10.2.10 EXT_DECL const TRDP_VERSION_T* tlc_getVersion (void)

Return version.

Return pointer to version structure

Return values:

const [TRDP_VERSION_T](#)

Return pointer to version structure

Return values:

[TRDP_VERSION_T](#)

5.10.2.11 EXT_DECL const CHAR8* tlc_getVersionString (void)

Return a human readable version representation.

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

Return values:

const string

5.10.2.12 EXT_DECL TRDP_ERR_T tlc_init (const TRDP_PRINT_DBG_T *pPrintDebugString*, const TRDP_MEM_CONFIG_T **pMemConfig*)

Initialize the TRDP stack.

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

Parameters:

← *pPrintDebugString* Pointer to debug print function

← *pMemConfig* Pointer to memory configuration

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR memory allocation failed

TRDP_PARAM_ERR initialization error

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

Parameters:

← *pPrintDebugString* Pointer to debug print function

← *pMemConfig* Pointer to memory configuration

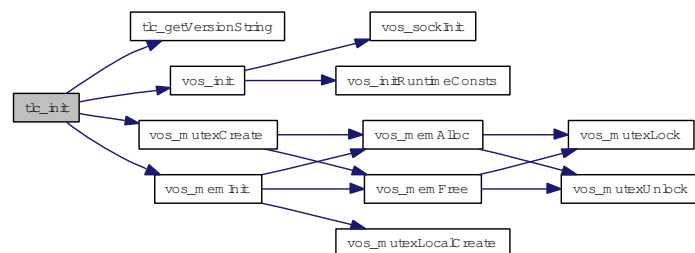
Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR memory allocation failed

TRDP_PARAM_ERR initialization error

Here is the call graph for this function:



5.10.2.13 `EXT_DECL TRDP_ERR_T tlc_openSession (TRDP_APP_SESSION_T * pAppHandle, TRDP_IP_ADDR_T ownIpAddr, TRDP_IP_ADDR_T leaderIpAddr, const TRDP_MARSHALL_CONFIG_T * pMarshall, const TRDP_PD_CONFIG_T * pPdDefault, const TRDP_MD_CONFIG_T * pMdDefault, const TRDP_PROCESS_CONFIG_T * pProcessConfig)`

Open a session with the TRDP stack.

`tlc_openSession` 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 multihoming systems, if zero, the default interface / IP will be used.
- ← *leaderIpAddr* IP address of redundancy leader
- ← *pMarshall* Pointer to marshalling configuration
- ← *pPdDefault* Pointer to default PD configuration
- ← *pMdDefault* Pointer to default MD configuration
- ← *pProcessConfig* Pointer to process configuration only option parameter is used here to define session behavior all other parameters are only used to feed statistics

Return values:

- TRDP_NO_ERR* no error
- TRDP_INIT_ERR* not yet initied
- TRDP_PARAM_ERR* parameter error
- TRDP SOCK_ERR* socket error

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

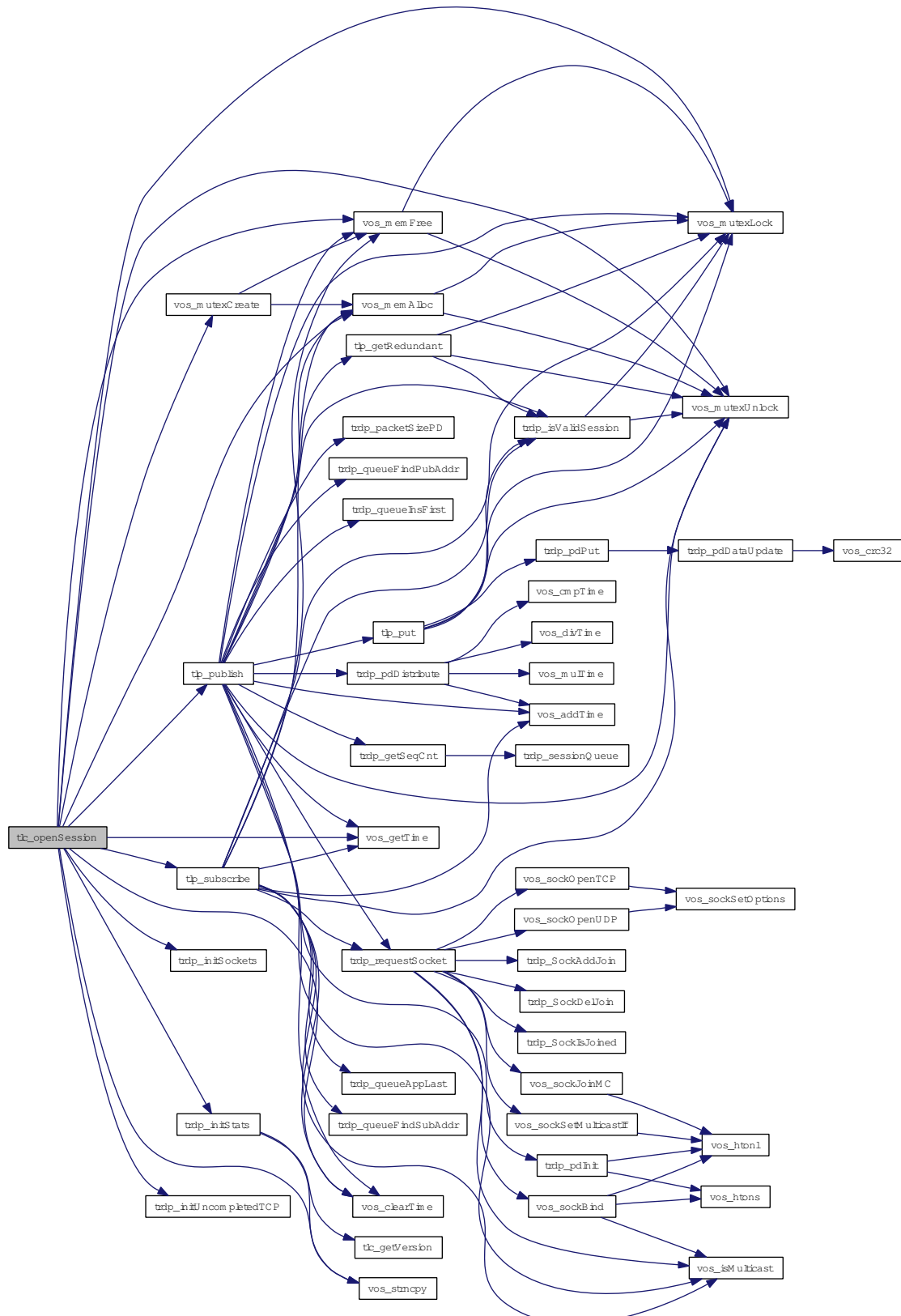
Parameters:

- *pAppHandle* A handle for further calls to the trdp stack
- ← *ownIpAddr* Own IP address, can be different for each process in multihoming systems, if zero, the default interface / IP will be used.
- ← *leaderIpAddr* IP address of redundancy leader
- ← *pMarshall* Pointer to marshalling configuration
- ← *pPdDefault* Pointer to default PD configuration
- ← *pMdDefault* Pointer to default MD configuration
- ← *pProcessConfig* Pointer to process configuration only option parameter is used here to define session behavior all other parameters are only used to feed statistics

Return values:

- TRDP_NO_ERR* no error
- TRDP_INIT_ERR* not yet initied
- TRDP_PARAM_ERR* parameter error
- TRDP SOCK_ERR* socket error

Here is the call graph for this function:



5.10.2.14 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

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_openSession

← *pRfds* pointer to set of ready descriptors

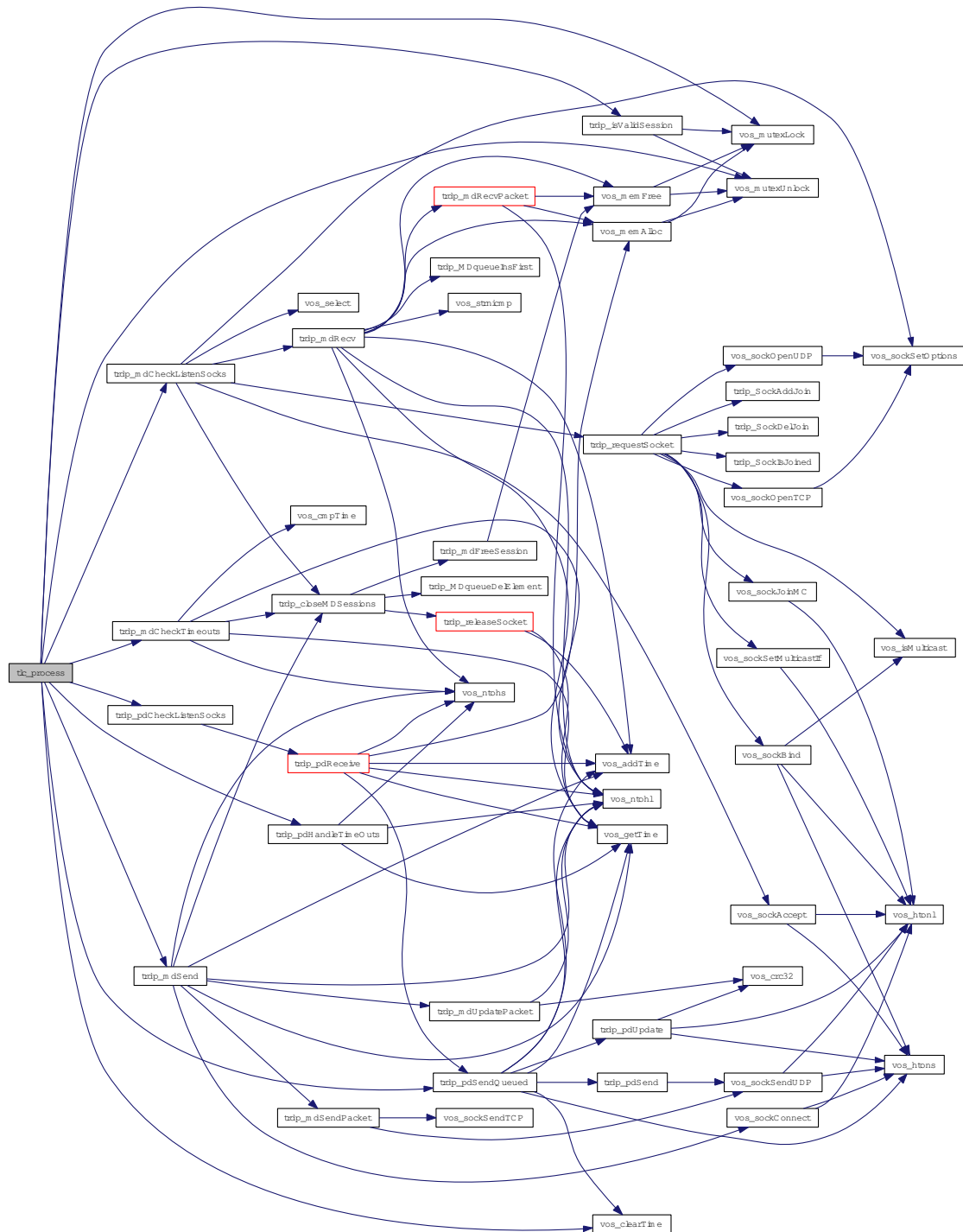
↔ *pCount* pointer to number of ready descriptors

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.15 EXT_DECL TRDP_ERR_T tlc_reinitSession (TRDP_APP_SESSION_T *appHandle*)

Re-Initialize.

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

Parameters:

← *appHandle* The handle returned by tlc_openSession

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

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

Parameters:

← *appHandle* The handle returned by tlc_openSession

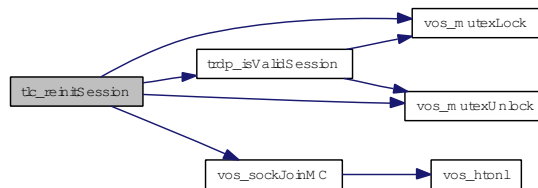
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Here is the call graph for this function:



5.10.2.16 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

Parameters:

← *appHandle* the handle returned by tlc_openSession

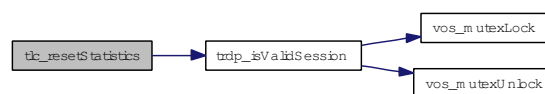
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.10.2.17 EXT_DECL TRDP_ERR_T tlc_setTopoCount (TRDP_APP_SESSION_T *appHandle*, 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:

← *appHandle* the handle returned by tlc_openSession

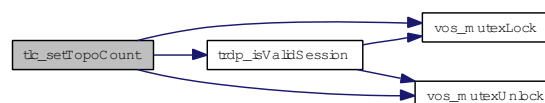
← *topoCount* New topoCount value

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.18 EXT_DECL TRDP_ERR_T tlc_terminate (void)

Un-Initialize.

Clean up and close all sessions. Mainly used for debugging/test runs. No further calls to library allowed

Return values:

TRDP_NO_ERR no error

Clean up and close all sessions. Mainly used for debugging/test runs. No further calls to library allowed

Return values:

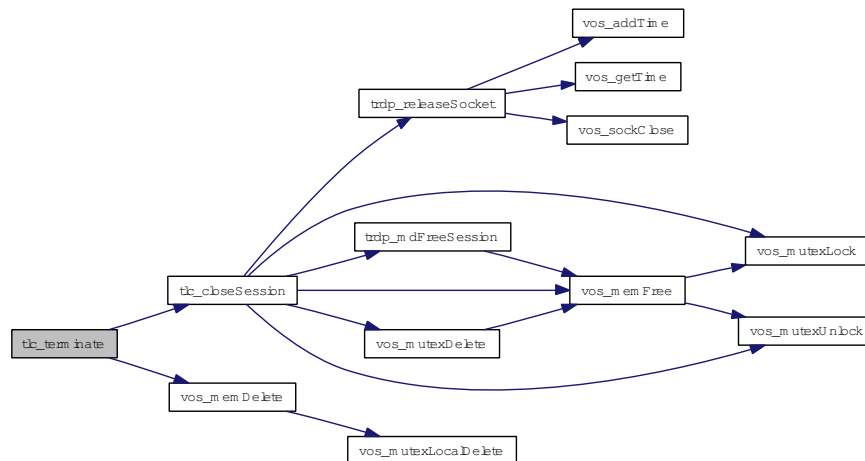
TRDP_NO_ERR no error

TRDP_INIT_ERR no error

TRDP_MEM_ERR TrafficStore nothing

TRDP_MUTEX_ERR TrafficStore mutex err

Here is the call graph for this function:



5.10.2.19 EXT_DECL TRDP_ERR_T tlm_abortSession (TRDP_APP_SESSION_T appHandle, const TRDP_UUID_T * pSessionId)

Cancel an open session.

Abort an open session; any pending messages will be dropped

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

5.10.2.20 `EXT_DECL TRDP_ERR_T tlm_addListener (TRDP_APP_SESSION_T appHandle, TRDP_LIS_T * pListenHandle, const void * pUserRef, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T mcDestIpAddr, 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
- ← *mcDestIpAddr* multicast group to listen on
- ← *pktFlags* OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_MARSHALL, TRDP_FLAGS_TCP
- ← *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_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 sourceURI, 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_DEFAULT

- ← *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
- ← *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_NO_SESSION_ERR* no such session
- TRDP_NOINIT_ERR* handle invalid

5.10.2.22 EXT_DECL TRDP_ERR_T tlm_delListener (TRDP_APP_SESSION_T *appHandle*, TRDP_LIS_T *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

5.10.2.23 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_DEFAULT, TRDP_FLAGS_MARSHALL, TRDP_FLAGS_TCP
- ← *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

5.10.2.24 **EXT_DECL** TRDP_ERR_T tlm_reply (TRDP_APP_SESSION_T *appHandle*, void * *pUserRef*, const TRDP_UUID_T * *pSessionId*, UINT32 *topoCount*, UINT32 *comId*, TRDP_IP_ADDR_T *srcIpAddr*, TRDP_IP_ADDR_T *destIpAddr*, TRDP_FLAGS_T *pktFlags*, UINT16 *userStatus*, const TRDP_SEND_PARAM_T * *pSendParam*, const UINT8 * *pData*, UINT32 *dataSize*, const TRDP_URI_USER_T *sourceURI*, 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
- ← *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* OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_MARSHALL
- ← *userStatus* Info for requester about application errors
- ← *pSendParam* pointer to send parameters, NULL to use default send parameters
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data
- ← *sourceURI* 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

5.10.2.25 `EXT_DECL TRDP_ERR_T tlm_replyErr (TRDP_APP_SESSION_T appHandle, const TRDP_UUID_T *pSessionId, UINT32 topoCount, UINT32 comId, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_REPLY_STATUS_T replyState, const TRDP_SEND_PARAM_T *pSendParam, const TRDP_URI_USER_T sourceURI, const TRDP_URI_USER_T destURI)`

Send a MD error reply message.

Send a MD error reply message after receiving an request

Parameters:

- ← *appHandle* the handle returned by `tlc_init`
- ← *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
- ← *replyState* Info for requester about stack errors
- ← *pSendParam* Pointer to send parameters, NULL to use default send parameters
- ← *sourceURI* 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

5.10.2.26 `EXT_DECL TRDP_ERR_T tlm_replyQuery (TRDP_APP_SESSION_T appHandle, void *pUserRef, const TRDP_UUID_T *pSessionId, UINT32 topoCount, UINT32 comId, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, UINT16 userStatus, UINT32 confirmTimeout, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, const TRDP_URI_USER_T sourceURI, const TRDP_URI_USER_T destURI)`

Send a MD reply message.

Send a MD reply message after receiving a request and ask for confirmation.

Parameters:

- ← *appHandle* the handle returned by `tlc_init`
- ← *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* OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_MARSHALL
- ← *userStatus* Info for requester about application errors
- ← *confirmTimeout* timeout for confirmation
- ← *pSendParam* Pointer to send parameters, NULL to use default send parameters
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data
- ← *sourceURI* 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

5.10.2.27 **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 numReplies, UINT32 replyTimeout, 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 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_DEFAULT, TRDP_FLAGS_MARSHALL, TRDP_FLAGS_TCP
- ← *numReplies* number of expected replies, 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

- ← *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

5.10.2.28 `EXT_DECL TRDP_ERR_T tlp_get (TRDP_APP_SESSION_T appHandle,
TRDP_SUB_T subHandle, 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
- ↔ *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
- TRDP_COMID_ERR* ComID not found when marshalling

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

Parameters:

- ← *appHandle* the handle returned by `tlc_openSession`
- ← *subHandle* the handle returned by subscription
- ↔ *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

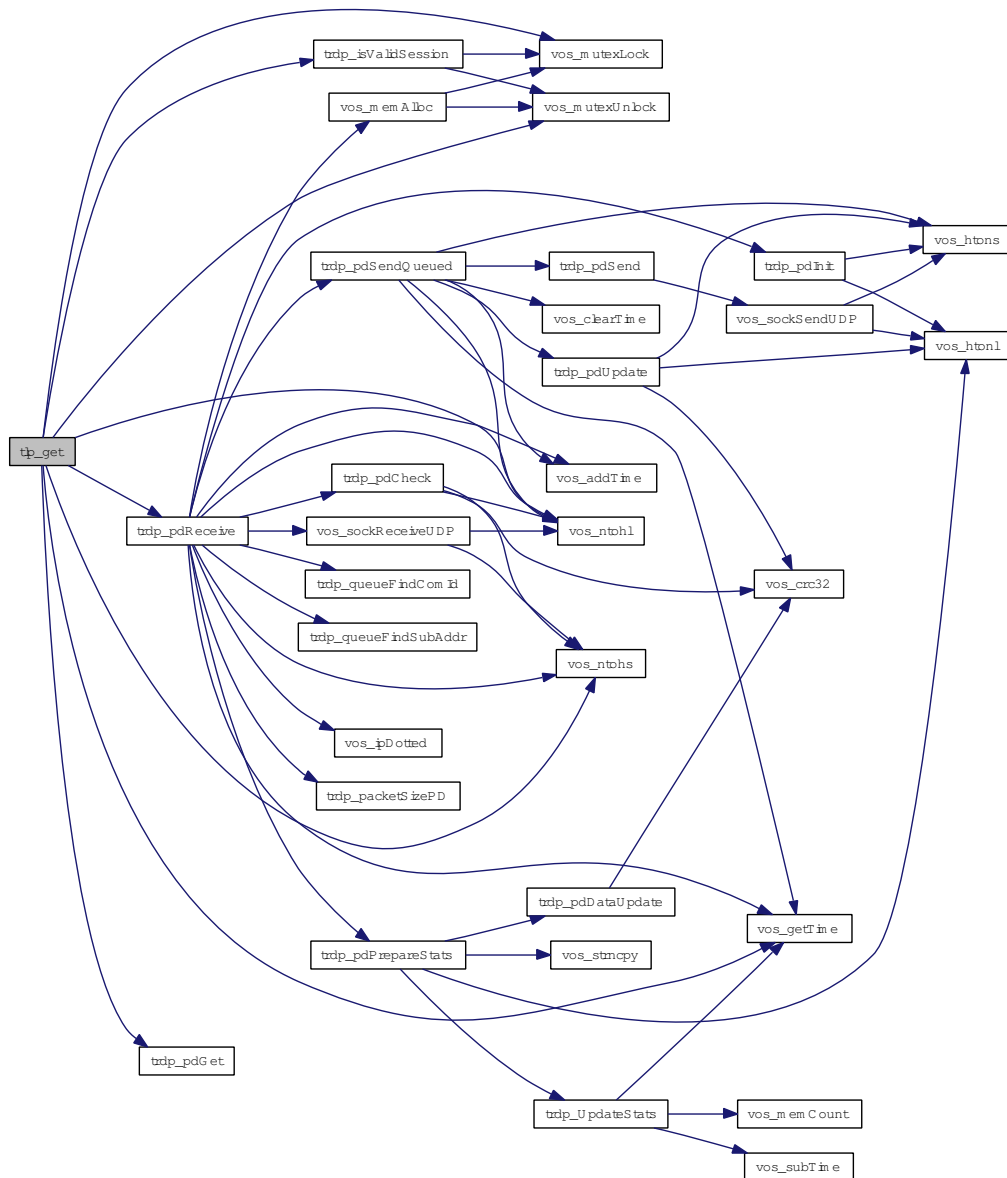
TRDP_SUB_ERR not subscribed

TRDP_TIMEOUT_ERR packet timed out

TRDP_NOINIT_ERR handle invalid

TRDP_COMID_ERR ComID not found when marshalling

Here is the call graph for this function:



5.10.2.29 EXT_DECL TRDP_ERR_T trdp_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

Only the status of the first redundancy group entry is returned will be returned!

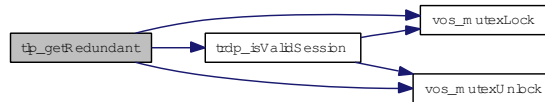
Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *redId* will be returned for all ComID's with the given redId
- ↔ *pLeader* TRUE if we're sending this redundancy group (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:



5.10.2.30 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*)

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 (≥ 10 ms) in usec
- ← *redId* 0 - Non-redundant, > 0 valid redundancy group
- ← *pktFlags* OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_NONE, 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

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

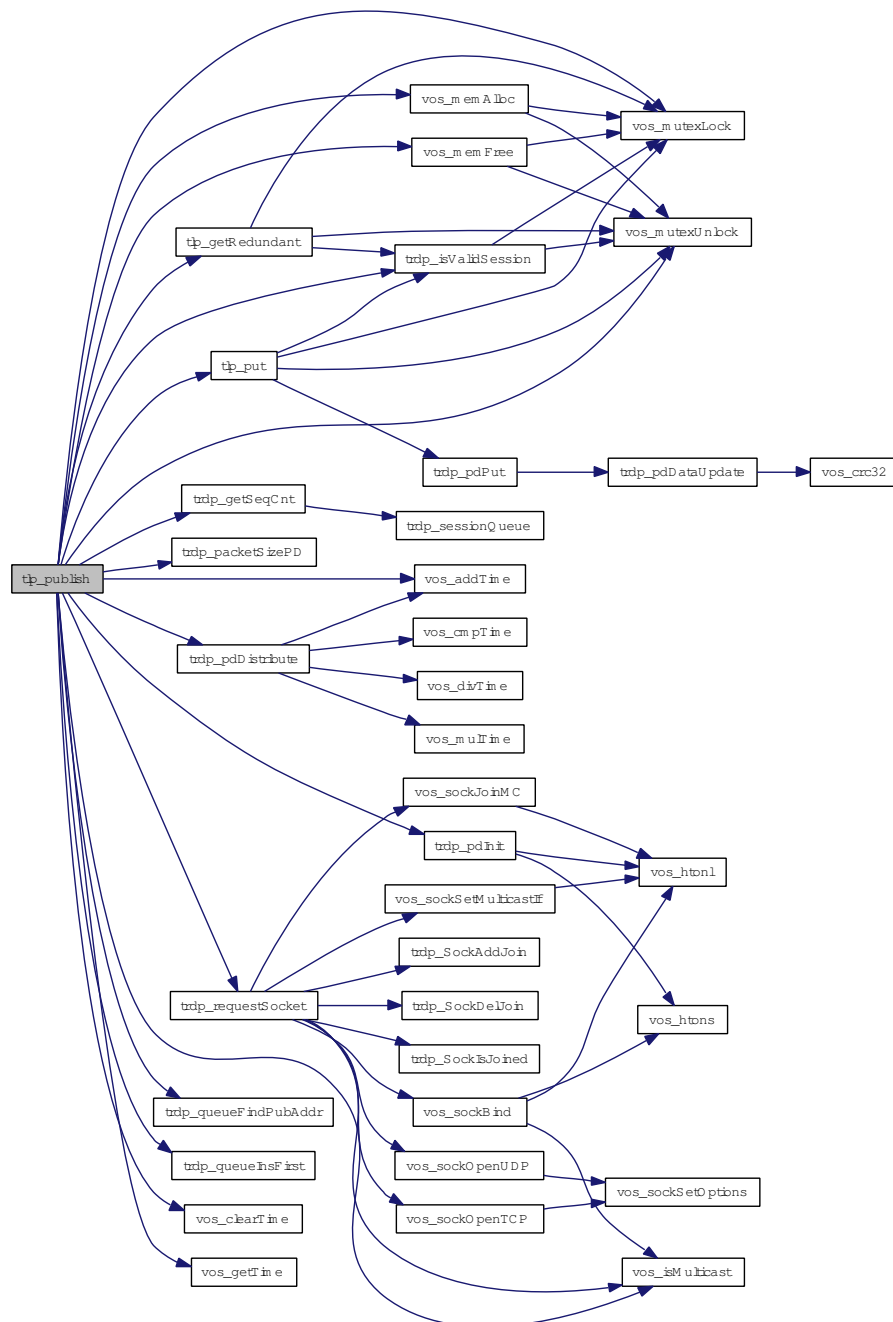
Parameters:

- ← *appHandle* the handle returned by tlc_openSession
- *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 (≥ 10 ms) in usec, 0 if PD PULL
- ← *redId* 0 - Non-redundant, > 0 valid redundancy group
- ← *pktFlags* OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_NONE, 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

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:



5.10.2.31 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 on uninitialized parameter or changed dataSize compared to published one
- TRDP_PUB_ERR** not published
- TRDP_NOINIT_ERR** handle invalid
- TRDP_COMID_ERR** ComID not found when marshalling

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

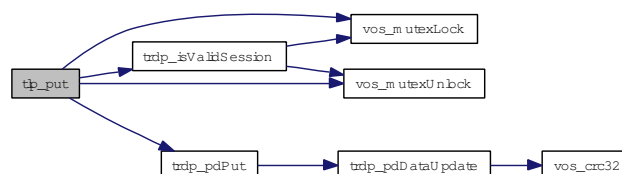
Parameters:

- ← *appHandle* the handle returned by tlc_openSession
- ← *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 on uninitialized parameter or changed dataSize compared to published one
- TRDP_NOPUB_ERR** not published
- TRDP_NOINIT_ERR** handle invalid
- TRDP_COMID_ERR** ComID not found when marshalling

Here is the call graph for this function:



5.10.2.32 **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*)

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: `TTRDP_FLAGS_DEFAULT`, `TRDP_FLAGS_NONE`, `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

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

Send a PD request message

Parameters:

- ← *appHandle* the handle returned by `tlc_openSession`
- ← *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* OPTION: `TRDP_FLAGS_DEFAULT`, `TRDP_FLAGS_NONE`, `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

Return values:

TRDP_NO_ERR no error

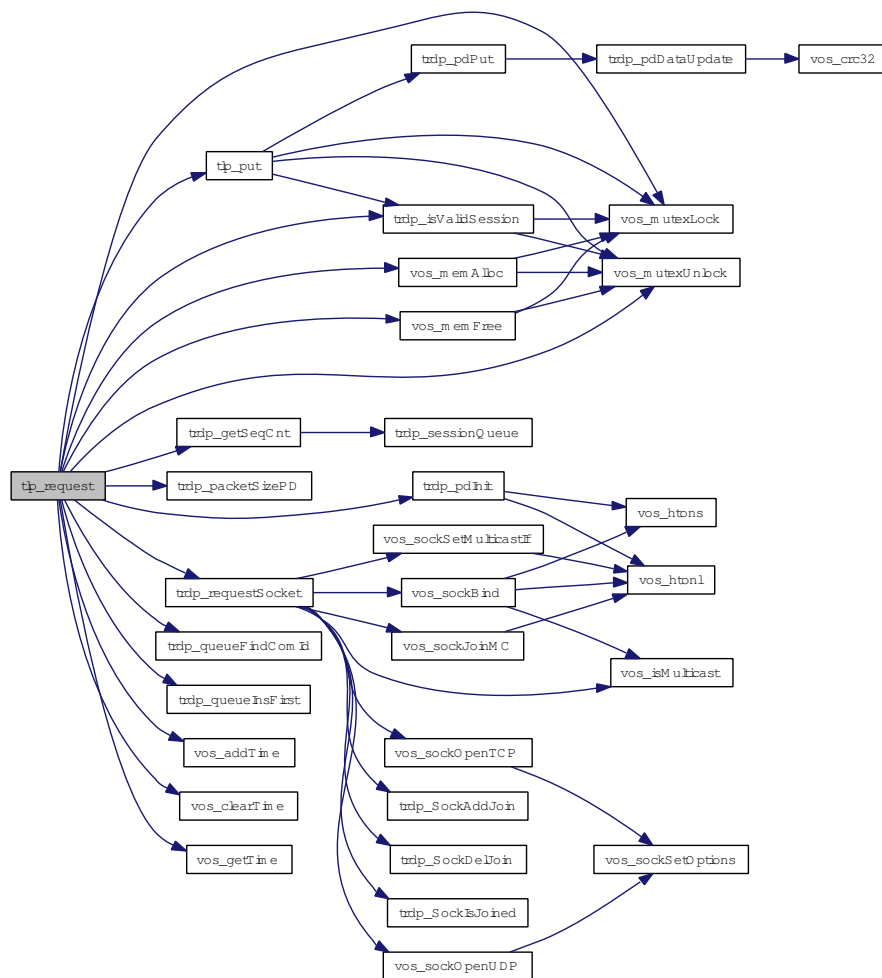
TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

TRDP_NOSUB_ERR no matching subscription found

Here is the call graph for this function:



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

Do not send redundant PD's 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

Do not send redundant PD's 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:



5.10.2.34 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*, TRDP_FLAGS_T *pktFlags*, 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
- ← *pktFlags* OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_NONE, TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *timeout* timeout (≥ 10 ms) in usec
- ← *toBehavior* OPTION: TRDP_TO_DEFAULT, TRDP_TO_SET_TO_ZERO, TRDP_TO_KEEP_LAST_VALUE
- ← *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

Subscribe to a specific PD ComID and source IP.

Parameters:

- ← *appHandle* the handle returned by tlc_openSession
- *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.
- ← *pktFlags* OPTION: TRDP_FLAGS_DEFAULT, TRDP_FLAGS_NONE, TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *destIpAddr* IP address to join
- ← *timeout* timeout (≥ 10 ms) 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)

← *pubHandle* the handle returned by prepare

Return values:

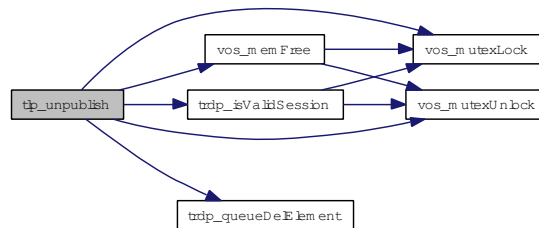
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOPUB_ERR not published

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.10.2.36 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

Unsubscribe to a specific PD ComID

Parameters:

← *appHandle* the handle returned by `tlc_openSession`

← *subHandle* the handle returned by subscription

Return values:

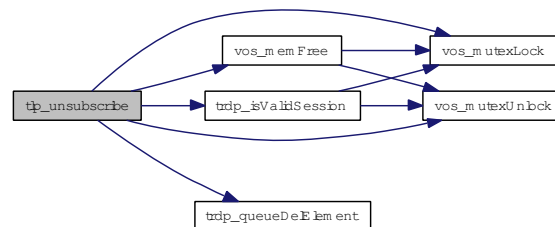
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOSUB_ERR not subscribed

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:

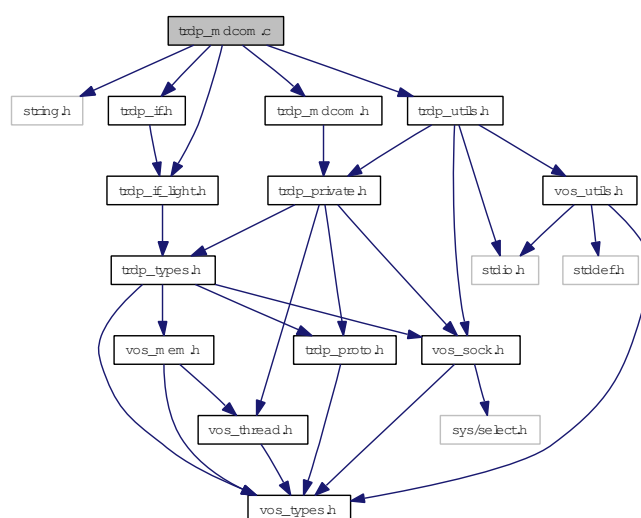


5.11 trdp_mdcom.c File Reference

Functions for MD communication.

```
#include <string.h>
#include "trdp_if_light.h"
#include "trdp_if.h"
#include "trdp_utils.h"
#include "trdp_mdcom.h"
```

Include dependency graph for trdp_mdcom.c:



Functions

- [TRDP_ERR_T trdp_getTCPSocket](#) ([TRDP_SESSION_PT](#) pSession)
Initialize the specific parameters for message data Open a listening socket.
- void [trdp_mdFreeSession](#) ([MD_ELE_T](#) *pMDSession)
Free memory of session.
- void [trdp_closeMDSessions](#) ([TRDP_SESSION_PT](#) appHandle, INT32 socketIndex, INT32 newSocket, BOOL checkAllSockets)
Close and free any session marked as dead.
- void [trdp_mdSetSessionTimeout](#) ([MD_ELE_T](#) *pMDSession, UINT32 usTimeOut)
set time out
- [TRDP_ERR_T trdp_mdCheck](#) ([TRDP_SESSION_PT](#) appHandle, [MD_HEADER_T](#) *pPacket, UINT32 packetSize, BOOL checkHeaderOnly)
Check for incoming md packet.
- void [trdp_mdUpdatePacket](#) ([MD_ELE_T](#) *pElement)

Update the header values.

- [TRDP_ERR_T trdp_mdSendPacket](#) (INT32 pdSock, UINT32 port, [MD_ELE_T](#) *pElement)

Send MD packet.

- [TRDP_ERR_T trdp_mdRecvPacket](#) ([TRDP_SESSION_PT](#) appHandle, INT32 mdSock, [MD_ELE_T](#) *pElement)

Receive MD packet.

- [TRDP_ERR_T trdp_mdRecv](#) ([TRDP_SESSION_PT](#) appHandle, UINT32 sockIndex)

Receiving MD messages Read the receive socket for arriving MDs, copy the packet to a new MD_ELE_T Check for protocol errors and dispatch to proper receive queue.

- [TRDP_ERR_T trdp_mdSend](#) ([TRDP_SESSION_PT](#) appHandle)

Sending MD messages Send the messages stored in the sendQueue Call user's callback if needed.

- void [trdp_mdCheckPending](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_FDS_T](#) *pFileDesc, INT32 *pNoDesc)

Check for pending packets, set FD if non blocking.

- void [trdp_mdCheckListenSocks](#) ([TRDP_SESSION_PT](#) appHandle, [TRDP_FDS_T](#) *pRfds, INT32 *pCount)

Checking receive connection requests and data Call user's callback if needed.

- void [trdp_mdCheckTimeouts](#) ([TRDP_SESSION_PT](#) appHandle)

Checking message data timeouts Call user's callback if needed.

5.11.1 Detailed Description

Functions for MD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Simone Pachera, FARsystems Gari Oiarbide, CAF Bernd Loehr, NewTec

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2013.

Id

[trdp_mdcom.c](#) 950 2013-06-13 13:51:41Z 97025

5.11.2 Function Documentation

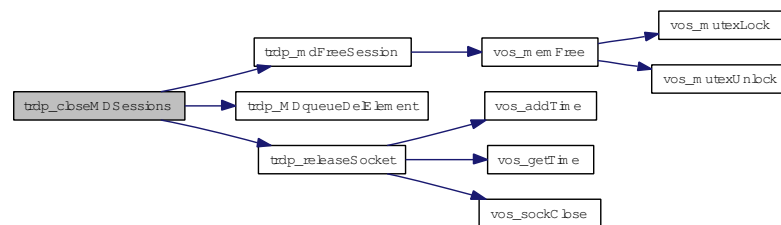
5.11.2.1 void trdp_closeMDSessions (TRDP_SESSION_PT *appHandle*, INT32 *socketIndex*, INT32 *newSocket*, BOOL *checkAllSockets*)

Close and free any session marked as dead.

Parameters:

- ← *appHandle* session pointer
- ← *socketIndex* the old socket position in the iface[]
- ← *newSocket* the new socket
- ← *checkAllSockets* check all the sockets that are waiting to be closed

Here is the call graph for this function:



5.11.2.2 TRDP_ERR_T trdp_getTCPSocket (TRDP_SESSION_PT *pSession*)

Initialize the specific parameters for message data Open a listening socket.

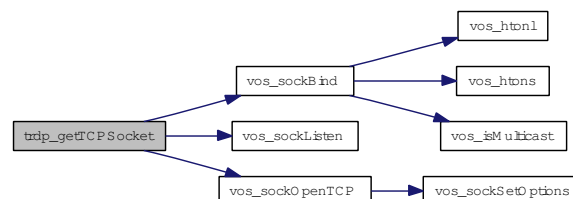
Parameters:

- ← *pSession* session parameters

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* initialization error

Here is the call graph for this function:



5.11.2.3 TRDP_ERR_T trdp_mdCheck (TRDP_SESSION_PT *appHandle*, MD_HEADER_T **pPacket*, UINT32 *packetSize*, BOOL *checkHeaderOnly*)

Check for incoming md packet.

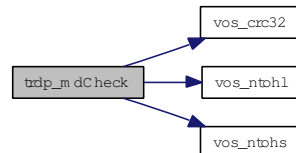
Parameters:

- ← *appHandle* session pointer
- ← *pPacket* pointer to the packet to check
- ← *packetSize* size of the packet
- ← *checkHeaderOnly* TRUE if data crc should not be checked

Return values:

- TRDP_NO_ERR* no error
- TRDP_TOPO_ERR*
- TRDP_WIRE_ERR*
- TRDP_CRC_ERR*

Here is the call graph for this function:



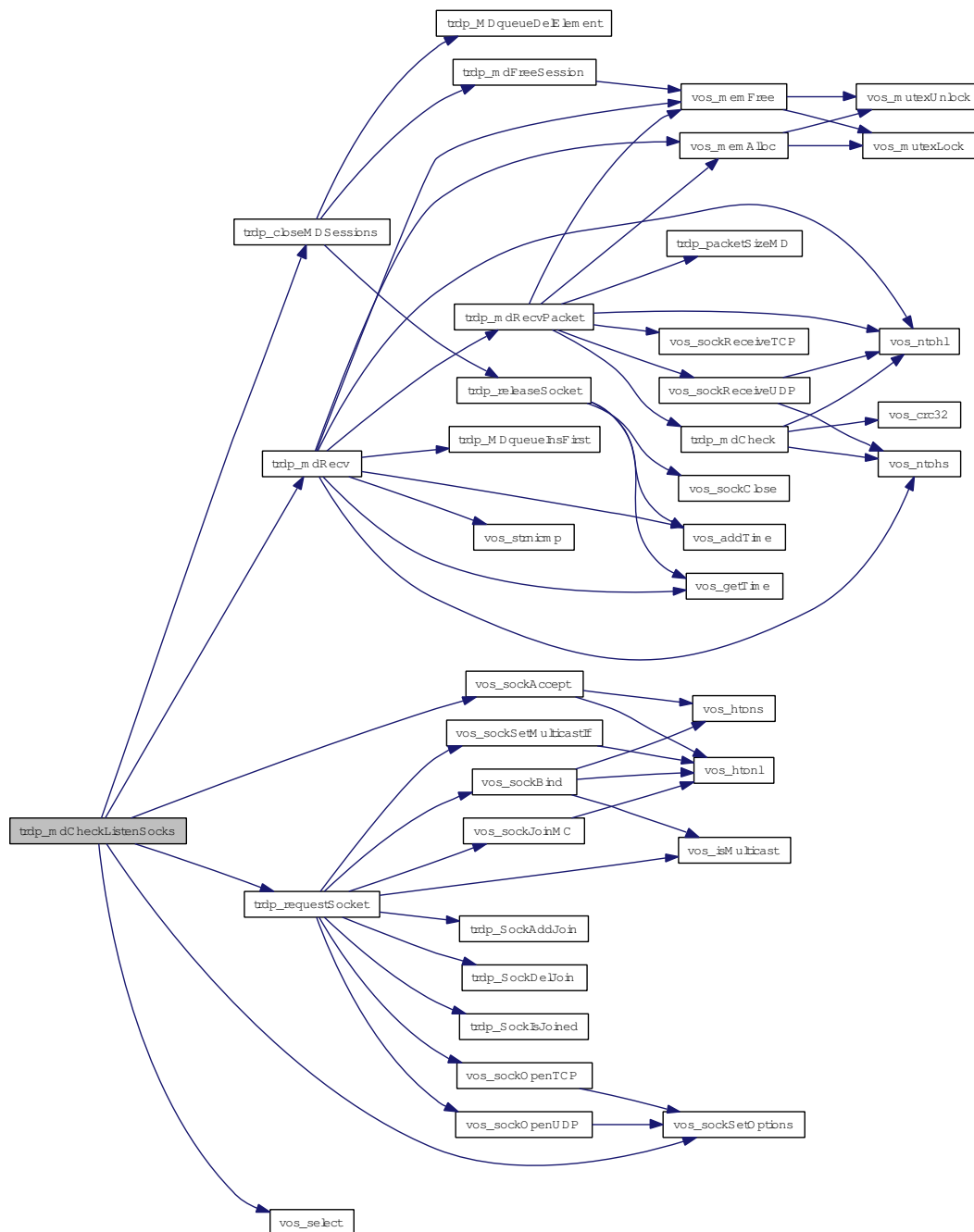
5.11.2.4 void trdp_mdCheckListenSocks (TRDP_SESSION_PT *appHandle*, TRDP_FDS_T **pRfds*, INT32 **pCount*)

Checking receive connection requests and data Call user's callback if needed.

Parameters:

- ← *appHandle* session pointer
- ← *pRfds* pointer to set of ready descriptors
- ↔ *pCount* pointer to number of ready descriptors

Here is the call graph for this function:



5.11.2.5 void trdp_mdCheckPending (TRDP_APP_SESSION_T *appHandle*, TRDP_FDS_T * *pFileDesc*, INT32 * *pNoDesc*)

Check for pending packets, set FD if non blocking.

Parameters:

- ← *appHandle* session pointer
- ↔ *pFileDesc* pointer to set of ready descriptors
- ↔ *pNoDesc* pointer to number of ready descriptors

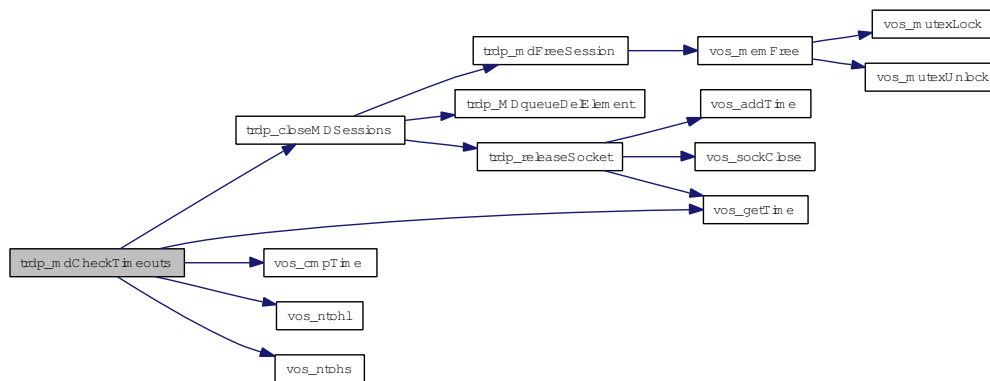
5.11.2.6 void trdp_mdCheckTimeouts (TRDP_SESSION_PT appHandle)

Checking message data timeouts Call user's callback if needed.

Parameters:

- ← *appHandle* session pointer

Here is the call graph for this function:

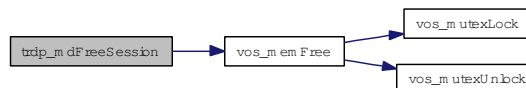
**5.11.2.7 void trdp_mdFreeSession (MD_ELEMENT * pMDSession)**

Free memory of session.

Parameters:

- ← *pMDSession* session pointer

Here is the call graph for this function:

**5.11.2.8 TRDP_ERR_T trdp_mdRecv (TRDP_SESSION_PT appHandle, UINT32 sockIndex)**

Receiving MD messages Read the receive socket for arriving MDs, copy the packet to a new MD_ELEMENT Check for protocol errors and dispatch to proper receive queue.

Call user's callback if needed

Parameters:

$\leftarrow appHandle$ session pointer

$\leftarrow sockIndex$ index of the socket to read from

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

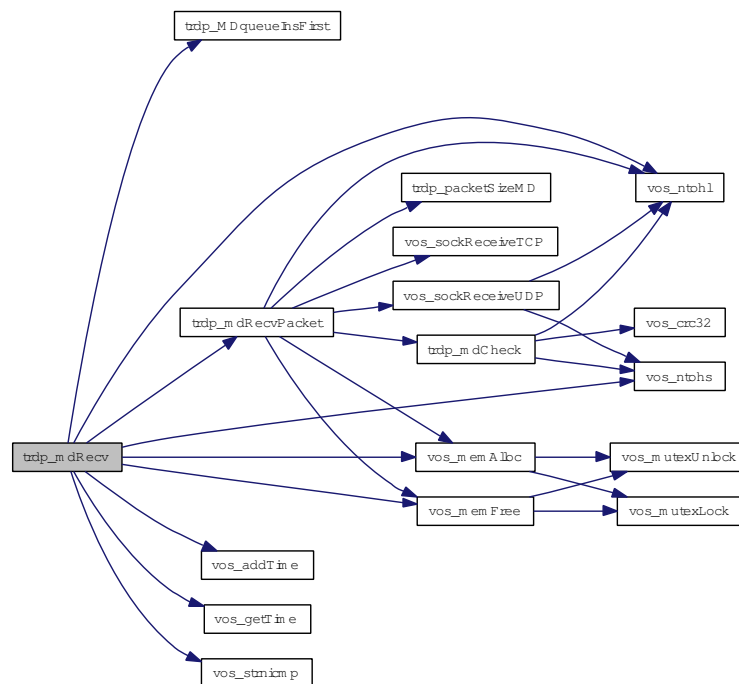
TRDP_WIRE_ERR protocol error (late packet, version mismatch)

TRDP_QUEUE_ERR not in queue

TRDP_CRC_ERR header checksum

TRDP_TOPOCOUNT_ERR invalid topocount

Here is the call graph for this function:



5.11.2.9 TRDP_ERR_T trdp_mdRecvPacket (TRDP_SESSION_PT *appHandle*, INT32 *mdSock*, MD_ELE_T * *pElement*)

Receive MD packet.

Parameters:

$\leftarrow appHandle$ session pointer

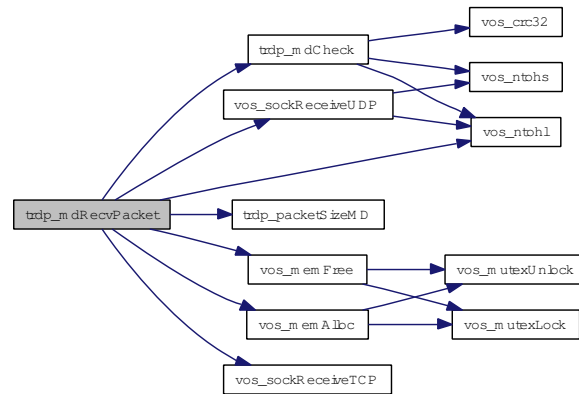
$\leftarrow mdSock$ socket descriptor

$\leftarrow pElement$ pointer to received packet

Return values:

`!= TRDP_NO_ERR` error

Here is the call graph for this function:



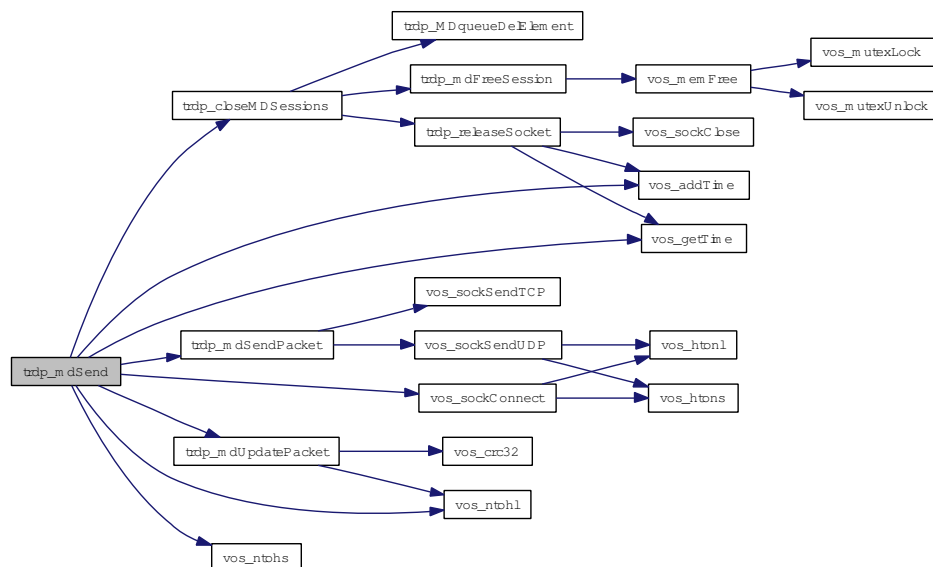
5.11.2.10 TRDP_ERR_T trdp_mdSend (TRDP_SESSION_PT *appHandle*)

Sending MD messages Send the messages stored in the sendQueue Call user's callback if needed.

Parameters:

← *appHandle* session pointer

Here is the call graph for this function:



5.11.2.11 TRDP_ERR_T trdp_mdSendPacket (INT32 *pdSock*, UINT32 *port*, MD_ELE_T **pElement*)

Send MD packet.

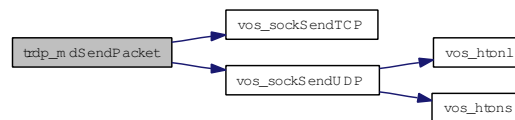
Parameters:

- ← *pdSock* socket descriptor
- ← *port* port on which to send
- ← *pElement* pointer to element to be sent

Return values:

- != NULL error

Here is the call graph for this function:



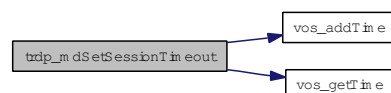
5.11.2.12 void trdp_mdSetSessionTimeout (MD_ELE_T **pMDSession*, UINT32 *usTimeout*)

set time out

Parameters:

- ← *pMDSession* session pointer
- ← *usTimeout* timeout in us

Here is the call graph for this function:



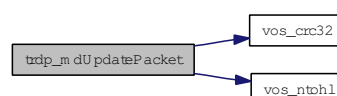
5.11.2.13 void trdp_mdUpdatePacket (MD_ELE_T **pElement*)

Update the header values.

Parameters:

- ← *pElement* pointer to the packet to update

Here is the call graph for this function:

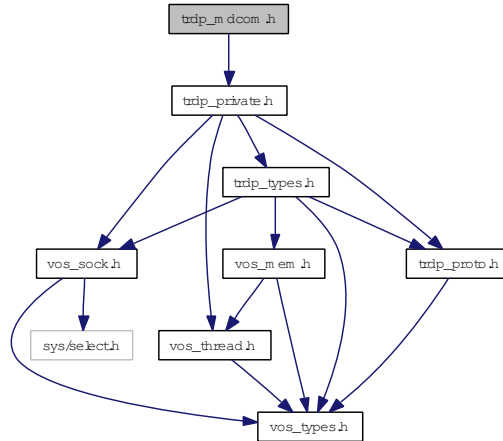


5.12 trdp_mdcom.h File Reference

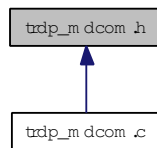
Functions for MD communication.

```
#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_getTCPSocket** (**TRDP_SESSION_PT** pSession)
Initialize the specific parameters for message data Open a listening socket.
- void **trdp_closeMDSessions** (**TRDP_SESSION_PT** appHandle, INT32 socketIndex, INT32 newSocket, BOOL checkAllSockets)
Close and free any session marked as dead.
- void **trdp_mdFreeSession** (**MD_ELE_T** *pMDSession)
Free memory of session.
- void **trdp_mdSetSessionTimeout** (**MD_ELE_T** *pMDSession, UINT32 usTimeOut)
set time out
- **TRDP_ERR_T** **trdp_mdSendPacket** (INT32 pdSock, UINT32 port, **MD_ELE_T** *pPacket)
Send MD packet.
- void **trdp_mdUpdatePacket** (**MD_ELE_T** *pPacket)

Update the header values.

- [TRDP_ERR_T trdp_mdRecv](#) ([TRDP_SESSION_PT](#) appHandle, [UINT32](#) sock)
Receiving MD messages Read the receive socket for arriving MDs, copy the packet to a new MD_ELEMENT Check for protocol errors and dispatch to proper receive queue.
- [TRDP_ERR_T trdp_mdSend](#) ([TRDP_SESSION_PT](#) appHandle)
Sending MD messages Send the messages stored in the sendQueue Call user's callback if needed.
- void [trdp_mdCheckPending](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_FDS_T](#) *pFileDesc, [INT32](#) *pNoDesc)
Check for pending packets, set FD if non blocking.
- void [trdp_mdCheckListenSocks](#) ([TRDP_SESSION_PT](#) appHandle, [TRDP_FDS_T](#) *pRfds, [INT32](#) *pCount)
Checking receive connection requests and data Call user's callback if needed.
- void [trdp_mdCheckTimeouts](#) ([TRDP_SESSION_PT](#) appHandle)
Checking message data timeouts Call user's callback if needed.

5.12.1 Detailed Description

Functions for MD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2013.

Id

[trdp_mdcom.h](#) 950 2013-06-13 13:51:41Z 97025

5.12.2 Function Documentation

5.12.2.1 void trdp_closeMDSessions ([TRDP_SESSION_PT](#) appHandle, [INT32](#) socketIndex, [INT32](#) newSocket, [BOOL](#) checkAllSockets)

Close and free any session marked as dead.

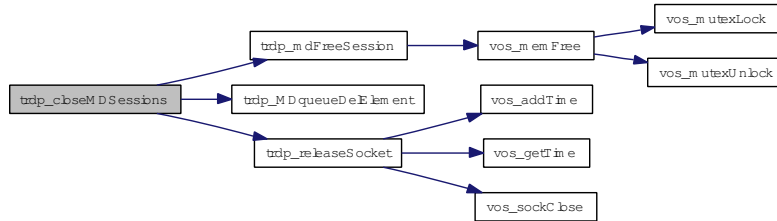
Parameters:

- ← *appHandle* session pointer
- ← *socketIndex* the old socket position in the iface[]

← *newSocket* the new socket

← *checkAllSockets* check all the sockets that are waiting to be closed

Here is the call graph for this function:



5.12.2.2 TRDP_ERR_T trdp_getTCPSocket (TRDP_SESSION_PT *pSession*)

Initialize the specific parameters for message data Open a listening socket.

Parameters:

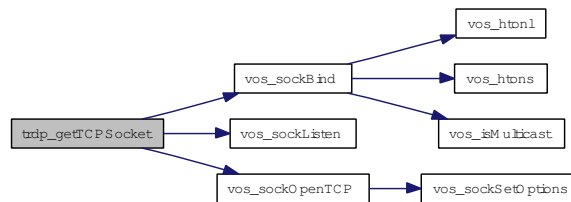
← *pSession* session parameters

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR initialization error

Here is the call graph for this function:



5.12.2.3 void trdp_mdCheckListenSocks (TRDP_SESSION_PT *appHandle*, TRDP_FDS_T **pRfds*, INT32 **pCount*)

Checking receive connection requests and data Call user's callback if needed.

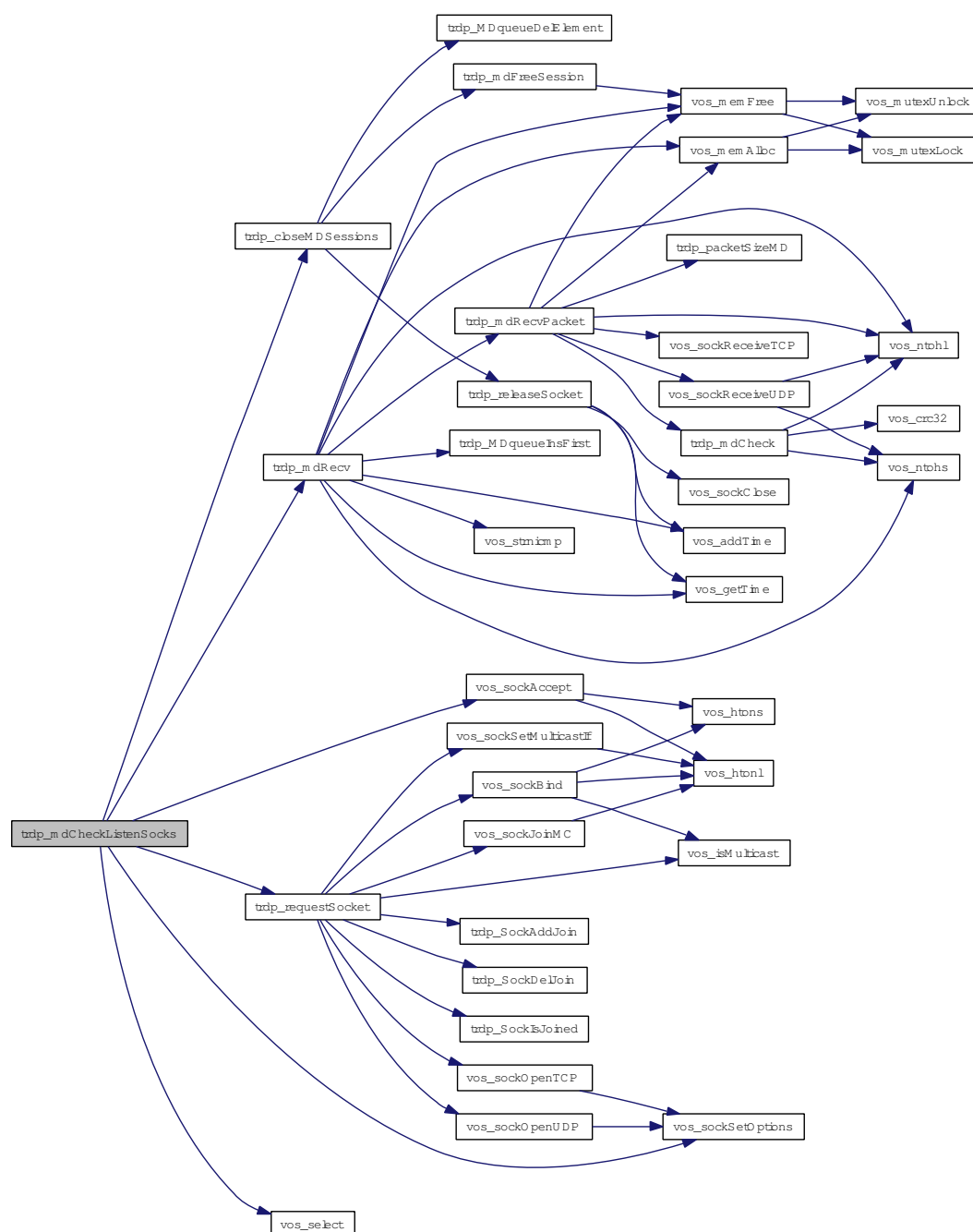
Parameters:

← *appHandle* session pointer

← *pRfds* pointer to set of ready descriptors

↔ *pCount* pointer to number of ready descriptors

Here is the call graph for this function:



5.12.2.4 void trdp_mdCheckPending (TRDP_APP_SESSION_T *appHandle*, TRDP_FDS_T * *pFileDesc*, INT32 * *pNoDesc*)

Check for pending packets, set FD if non blocking.

Parameters:

- ← *appHandle* session pointer
- ↔ *pFileDesc* pointer to set of ready descriptors
- ↔ *pNoDesc* pointer to number of ready descriptors

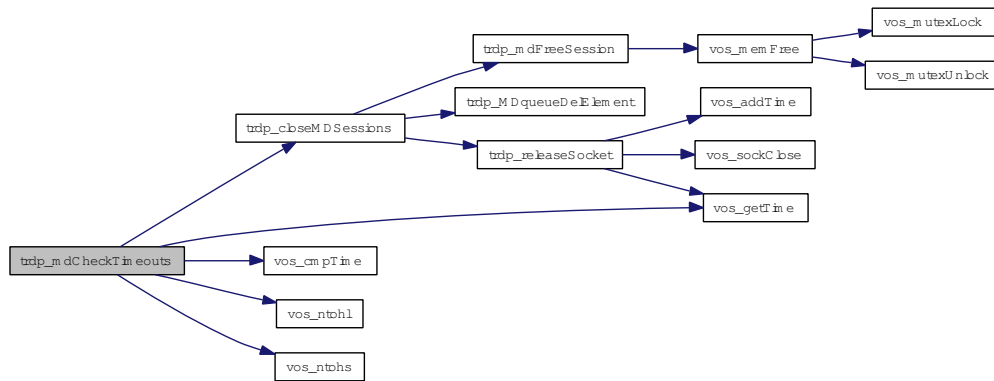
5.12.2.5 void trdp_mdCheckTimeouts (TRDP_SESSION_PT appHandle)

Checking message data timeouts Call user's callback if needed.

Parameters:

- ← *appHandle* session pointer

Here is the call graph for this function:

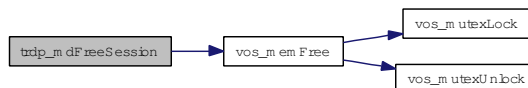
**5.12.2.6 void trdp_mdFreeSession (MD_ELEMENT * pMDSession)**

Free memory of session.

Parameters:

- ← *pMDSession* session pointer

Here is the call graph for this function:

**5.12.2.7 TRDP_ERR_T trdp_mdRecv (TRDP_SESSION_PT appHandle, UINT32 sockIndex)**

Receiving MD messages Read the receive socket for arriving MDs, copy the packet to a new MD_ELEMENT Check for protocol errors and dispatch to proper receive queue.

Call user's callback if needed

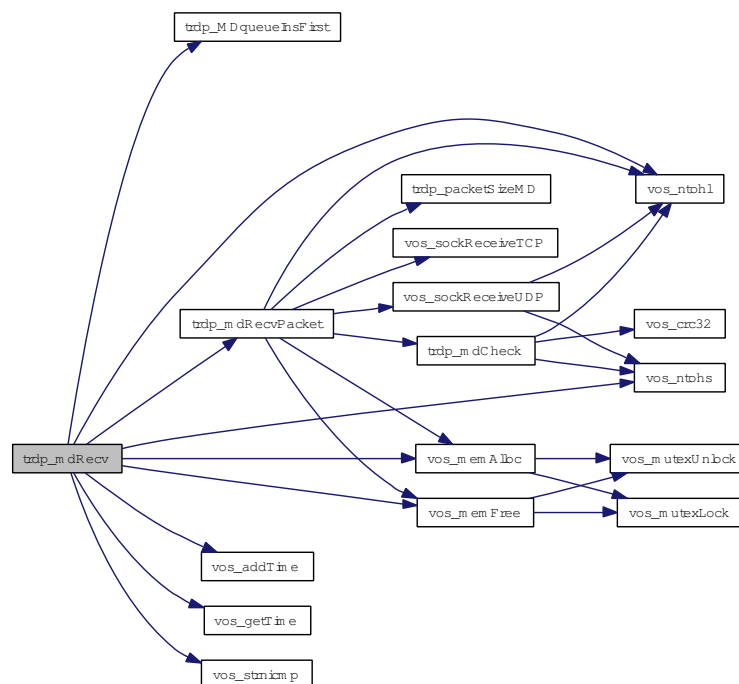
Parameters:

- ← *appHandle* session pointer
- ← *sockIndex* index of 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:

**5.12.2.8 TRDP_ERR_T trdp_mdSend (TRDP_SESSION_PT appHandle)**

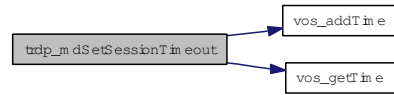
Sending MD messages Send the messages stored in the sendQueue Call user's callback if needed.

Parameters:

- ← *appHandle* session pointer

← *usTimeout* timeout in us

Here is the call graph for this function:



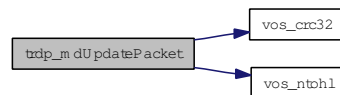
5.12.2.11 void trdp_mdUpdatePacket (MD_ELEMENT *pElement)

Update the header values.

Parameters:

← *pElement* pointer to the packet to update

Here is the call graph for this function:

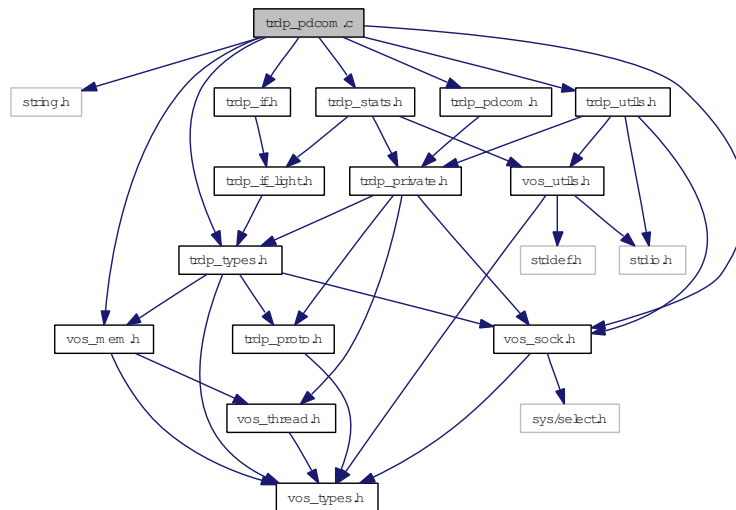


5.13 trdp_pdcom.c File Reference

Functions for PD communication.

```
#include <string.h>
#include "trdp_types.h"
#include "trdp_utils.h"
#include "trdp_pdcom.h"
#include "trdp_if.h"
#include "trdp_stats.h"
#include "vos_sock.h"
#include "vos_mem.h"
```

Include dependency graph for trdp_pdcom.c:



Functions

- void [trdp_pdInit](#) ([PD_ELE_T](#) *pPacket, [TRDP_MSG_T](#) type, UINT32 topoCount, UINT32 reply-ComId, UINT32 replyIpAddress)
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.
- void [trdp_pdDataUpdate](#) ([PD_ELE_T](#) *pPacket)
Add padding and update data CRC.
- [TRDP_ERR_T](#) [trdp_pdGet](#) ([PD_ELE_T](#) *pPacket, [TRDP_UNMARSHALL_T](#) unmarshall, void *refCon, const UINT8 *pData, UINT32 *pDataSize)
Copy data Set the header infos.

- [TRDP_ERR_T trdp_pdSendQueued](#) ([TRDP_SESSION_PT](#) appHandle)
Send all due PD messages.
- [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_pdCheckPending](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_FDS_T](#) *pFileDesc, INT32 *pNoDesc)
Check for pending packets, set FD if non blocking.
- void [trdp_pdHandleTimeOuts](#) ([TRDP_SESSION_PT](#) appHandle)
Check for time outs.
- [TRDP_ERR_T trdp_pdCheckListenSocks](#) ([TRDP_SESSION_PT](#) appHandle, [TRDP_FDS_T](#) *pRfds, INT32 *pCount)
Checking receive connection requests and data Call user's callback if needed.
- void [trdp_pdUpdate](#) ([PD_ELE_T](#) *pPacket)
Update the header values.
- [TRDP_ERR_T trdp_pdCheck](#) ([PD_HEADER_T](#) *pPacket, UINT32 packetSize)
Check if the PD header values and the CRCs are sane.
- [TRDP_ERR_T trdp_pdSend](#) (INT32 pdSock, [PD_ELE_T](#) *pPacket, UINT16 port)
Send one PD packet.
- [TRDP_ERR_T trdp_pdDistribute](#) ([PD_ELE_T](#) *pSndQueue)
Distribute send time of PD packets over time.

5.13.1 Detailed Description

Functions for PD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_pdcom.c](#) 950 2013-06-13 13:51:41Z 97025

BL 2013-04-09: ID 92: Pull request led to reset of push message type BL 2013-01-25: ID 20: Redundancy handling fixed

5.13.2 Function Documentation

5.13.2.1 TRDP_ERR_T trdp_pdCheck (PD_HEADER_T * *pPacket*, UINT32 *packetSize*)

Check if the PD header values and the CRCs are sane.

Parameters:

← *pPacket* pointer to the packet to check

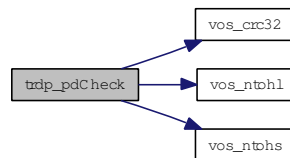
← *packetSize* max size to check

Return values:

TRDP_NO_ERR

TRDP_CRC_ERR

Here is the call graph for this function:



5.13.2.2 TRDP_ERR_T trdp_pdCheckListenSocks (TRDP_SESSION_PT *appHandle*, TRDP_FDS_T * *pRfds*, INT32 * *pCount*)

Checking receive connection requests and data Call user's callback if needed.

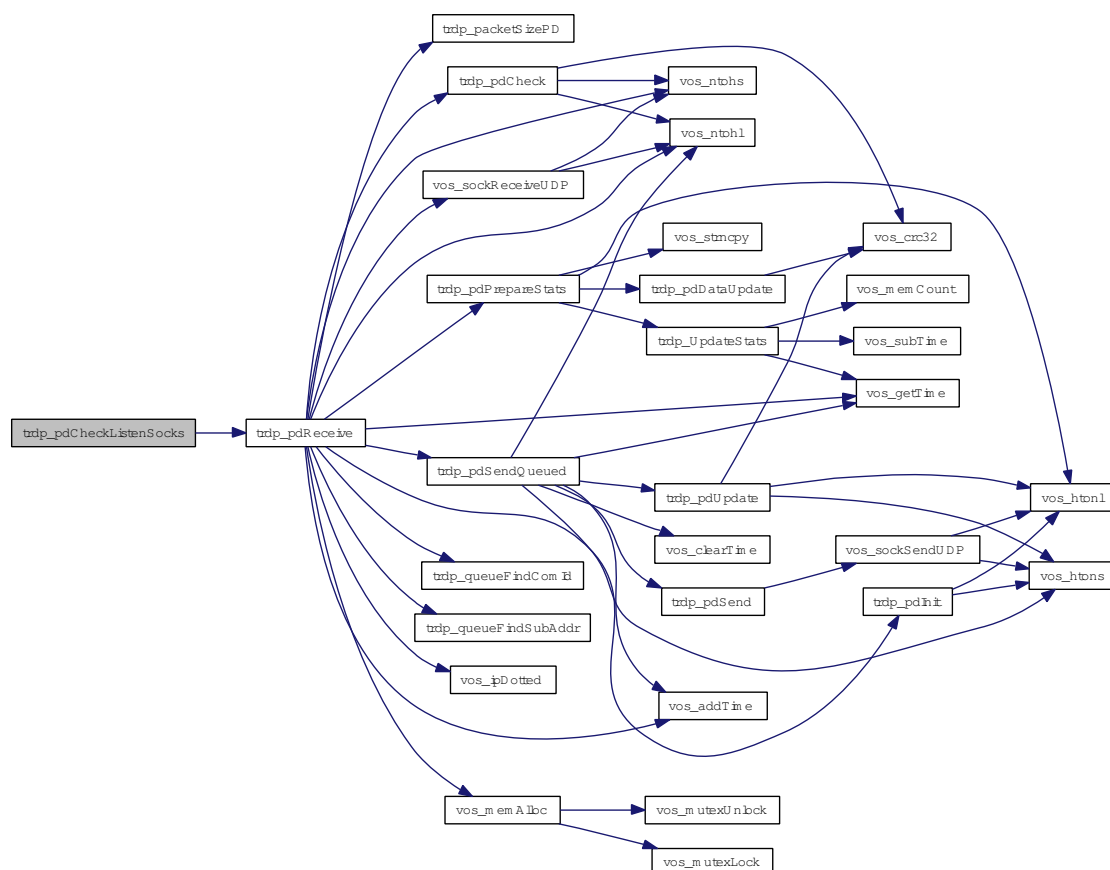
Parameters:

← *appHandle* session pointer

← *pRfds* pointer to set of ready descriptors

↔ *pCount* pointer to number of ready descriptors

Here is the call graph for this function:



5.13.2.3 void trdp_pdCheckPending (TRDP_APP_SESSION_T *appHandle*, TRDP_FDS_T * *pFileDesc*, INT32 * *pNoDesc*)

Check for pending packets, set FD if non blocking.

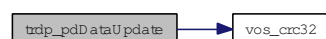
Parameters:

- ← *appHandle* session pointer
- ↔ *pFileDesc* pointer to set of ready descriptors
- ↔ *pNoDesc* pointer to number of ready descriptors

5.13.2.4 void trdp_pdDataUpdate (PD_ELE_T * *pPacket*)

Add padding and update data CRC.

Here is the call graph for this function:



5.13.2.5 TRDP_ERR_T trdp_pdDistribute (PD_ELE_T * pSndQueue)

Distribute send time of PD packets over time.

The duration of PD packets on a 100MBit/s network ranges from 3us to 150us max. Because a cyclic thread scheduling below 5ms would put a too heavy load on the system, and PD packets cannot get larger than 1436 (+ UDP header), we will not account for differences in packet size. Another factor is the differences in intervals for different packets: We should only change the starting times of the packets within 1/2 the interval time. Otherwise a late addition of packets could lead to timeouts of already queued packets. Scheduling will be computed based on the smallest interval time.

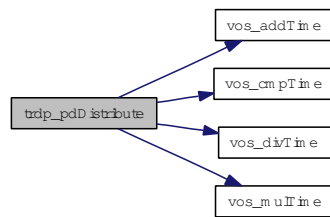
Parameters:

← *pSndQueue* pointer to send queue

Return values:

TRDP_NO_ERR

Here is the call graph for this function:



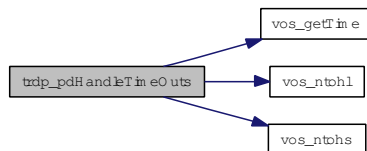
5.13.2.6 void trdp_pdHandleTimeOuts (TRDP_SESSION_PT appHandle)

Check for time outs.

Parameters:

← *appHandle* application handle

Here is the call graph for this function:



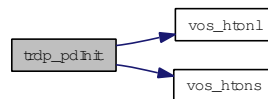
5.13.2.7 void trdp_pdInit (PD_ELE_T * pPacket, TRDP_MSG_T type, UINT32 topoCount, UINT32 replyComId, UINT32 replyIpAddress)

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
- ← *replyComId* Pull request comId
- ← *replyIpAddress* Pull request Ip

Here is the call graph for this function:



5.13.2.8 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, check if it is a PD Request (PULL). If it is an update, exchange the existing entry with the new one Call user's callback if needed

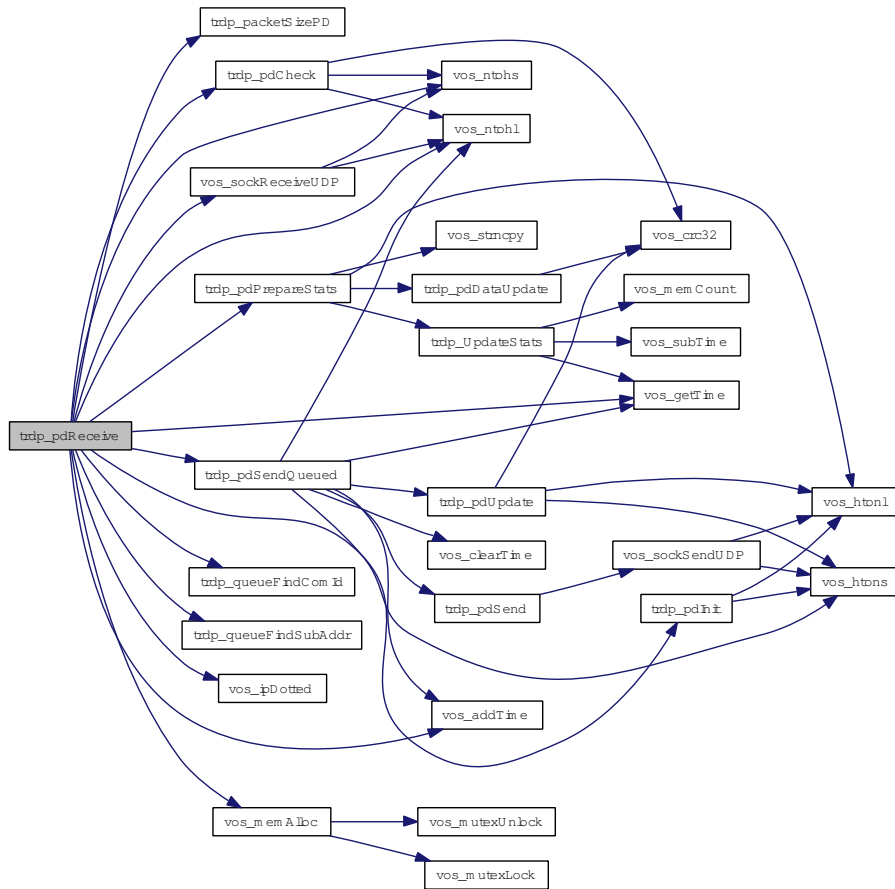
Parameters:

- ← *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:



5.13.2.9 TRDP_ERR_T trdp_pdSend (INT32 *pdSock*, PD_ELEMENT * *pPacket*, UINT16 *port*)

Send one PD packet.

Parameters:

- ← *pdSock* socket descriptor
- ← *pPacket* pointer to packet to be sent
- ← *port* port on which to send

Return values:

TRDP_NO_ERR
TRDP_IO_ERR

Here is the call graph for this function:



5.13.2.10 TRDP_ERR_T trdp_pdSendQueued (TRDP_SESSION_PT *appHandle*)

Send all due PD messages.

Parameters:

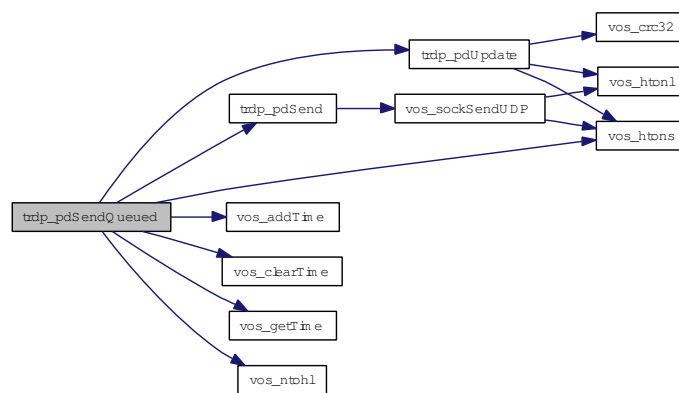
← *appHandle* session pointer

Return values:

TRDP_NO_ERR no error

TRDP_IO_ERR socket I/O error

Here is the call graph for this function:



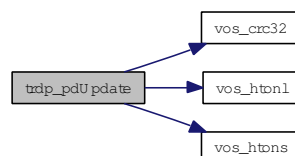
5.13.2.11 void trdp_pdUpdate (PD_ELE_T * *pPacket*)

Update the header values.

Parameters:

← *pPacket* pointer to the packet to update

Here is the call graph for this function:

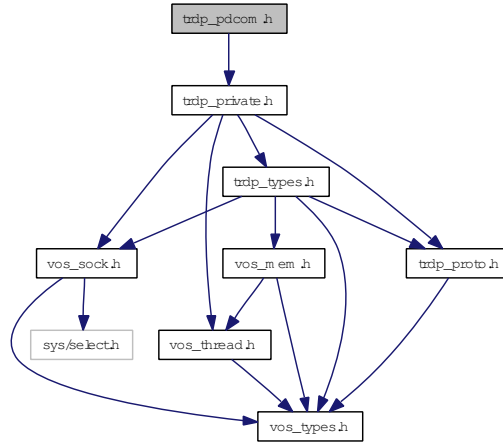


5.14 trdp_pdcom.h File Reference

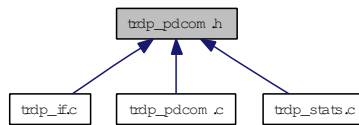
Functions for PD communication.

```
#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, UINT32 replyComId, UINT32 replyIpAddress)
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.
- void [trdp_pdDataUpdate](#) ([PD_ELE_T](#) *pPacket)
Add padding and update data CRC.
- [TRDP_ERR_T](#) [trdp_pdCheck](#) ([PD_HEADER_T](#) *pPacket, [UINT32](#) packetSize)
Check if the PD header values and the CRCs are sane.
- [TRDP_ERR_T](#) [trdp_pdSend](#) ([INT32](#) pdSock, [PD_ELE_T](#) *pPacket, [UINT16](#) port)

Send one PD packet.

- [TRDP_ERR_T trdp_pdGet](#) ([PD_ELE_T](#) *pPacket, [TRDP_UNMARSHALL_T](#) unmarshall, void *refCon, const [UINT8](#) *pData, [UINT32](#) *pDataSize)

Copy data Set the header infos.

- [TRDP_ERR_T trdp_pdSendQueued](#) ([TRDP_SESSION_PT](#) appHandle)

Send all due PD messages.

- [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.

- void [trdp_pdCheckPending](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_FDS_T](#) *pFileDesc, [INT32](#) *pNoDesc)

Check for pending packets, set FD if non blocking.

- void [trdp_pdHandleTimeOuts](#) ([TRDP_SESSION_PT](#) appHandle)

Check for time outs.

- [TRDP_ERR_T trdp_pdCheckListenSocks](#) ([TRDP_SESSION_PT](#) appHandle, [TRDP_FDS_T](#) *pRfds, [INT32](#) *pCount)

Checking receive connection requests and data Call user's callback if needed.

- [TRDP_ERR_T trdp_pdDistribute](#) ([PD_ELE_T](#) *pSndQueue)

Distribute send time of PD packets over time.

5.14.1 Detailed Description

Functions for PD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_pdcom.h](#) 950 2013-06-13 13:51:41Z 97025

5.14.2 Function Documentation

5.14.2.1 TRDP_ERR_T trdp_pdCheck (PD_HEADER_T * *pPacket*, UINT32 *packetSize*)

Check if the PD header values and the CRCs are sane.

Parameters:

← *pPacket* pointer to the packet to check

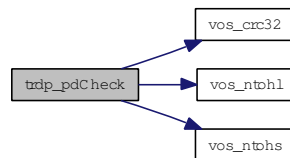
← *packetSize* max size to check

Return values:

TRDP_NO_ERR

TRDP_CRC_ERR

Here is the call graph for this function:



5.14.2.2 TRDP_ERR_T trdp_pdCheckListenSocks (TRDP_SESSION_PT *appHandle*, TRDP_FDS_T * *pRfds*, INT32 * *pCount*)

Checking receive connection requests and data Call user's callback if needed.

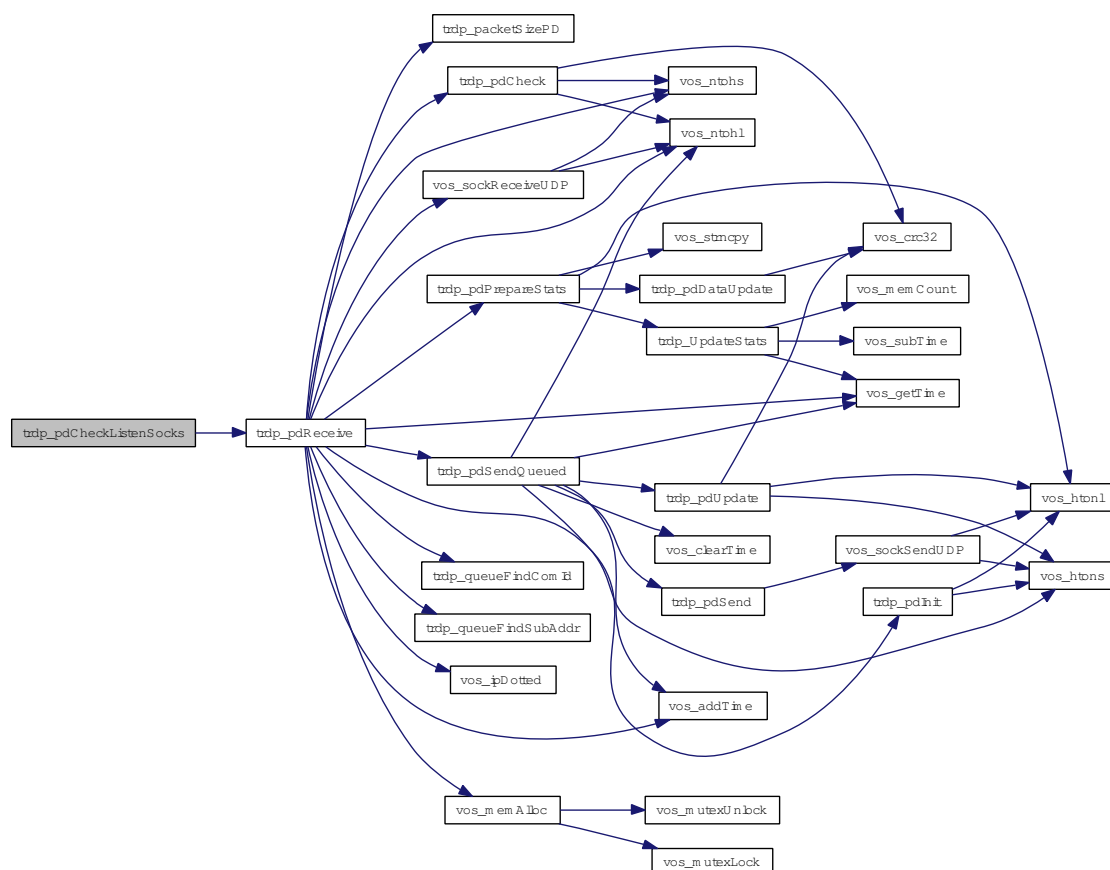
Parameters:

← *appHandle* session pointer

← *pRfds* pointer to set of ready descriptors

↔ *pCount* pointer to number of ready descriptors

Here is the call graph for this function:



5.14.2.3 void trdp_pdCheckPending (TRDP_APP_SESSION_T *appHandle*, TRDP_FDS_T * *pFileDesc*, INT32 * *pNoDesc*)

Check for pending packets, set FD if non blocking.

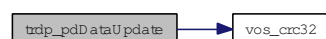
Parameters:

- ← *appHandle* session pointer
- ↔ *pFileDesc* pointer to set of ready descriptors
- ↔ *pNoDesc* pointer to number of ready descriptors

5.14.2.4 void trdp_pdDataUpdate (PD_ELE_T * *pPacket*)

Add padding and update data CRC.

Here is the call graph for this function:



5.14.2.5 TRDP_ERR_T trdp_pdDistribute (PD_ELE_T * pSndQueue)

Distribute send time of PD packets over time.

The duration of PD packets on a 100MBit/s network ranges from 3us to 150us max. Because a cyclic thread scheduling below 5ms would put a too heavy load on the system, and PD packets cannot get larger than 1436 (+ UDP header), we will not account for differences in packet size. Another factor is the differences in intervals for different packets: We should only change the starting times of the packets within 1/2 the interval time. Otherwise a late addition of packets could lead to timeouts of already queued packets. Scheduling will be computed based on the smallest interval time.

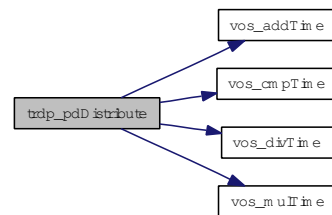
Parameters:

← *pSndQueue* pointer to send queue

Return values:

TRDP_NO_ERR

Here is the call graph for this function:



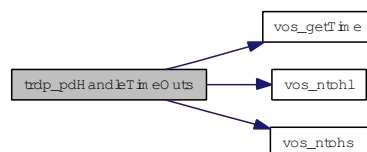
5.14.2.6 void trdp_pdHandleTimeOuts (TRDP_SESSION_PT appHandle)

Check for time outs.

Parameters:

← *appHandle* application handle

Here is the call graph for this function:



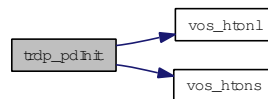
5.14.2.7 void trdp_pdInit (PD_ELE_T * pPacket, TRDP_MSG_T type, UINT32 topoCount, UINT32 replyComId, UINT32 replyIpAddress)

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
- ← *replyComId* Pull request comId
- ← *replyIpAddress* Pull request Ip

Here is the call graph for this function:



5.14.2.8 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, check if it is a PD Request (PULL). 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

5.14.2.10 TRDP_ERR_T trdp_pdSendQueued (TRDP_SESSION_PT *appHandle*)

Send all due PD messages.

Parameters:

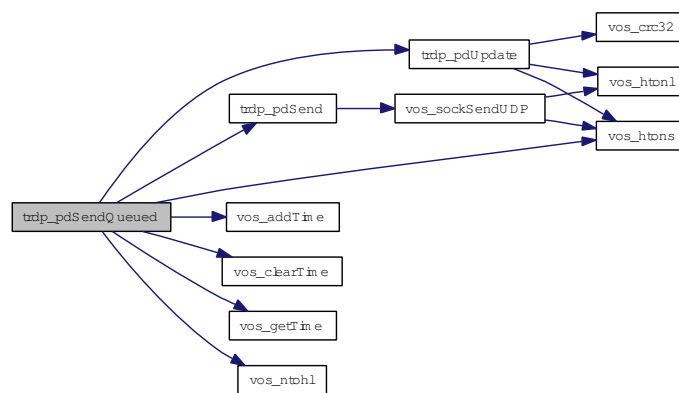
← *appHandle* session pointer

Return values:

TRDP_NO_ERR no error

TRDP_IO_ERR socket I/O error

Here is the call graph for this function:



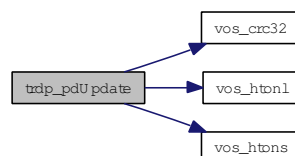
5.14.2.11 void trdp_pdUpdate (PD_ELE_T * *pPacket*)

Update the header values.

Parameters:

← *pPacket* pointer to the packet to update

Here is the call graph for this function:

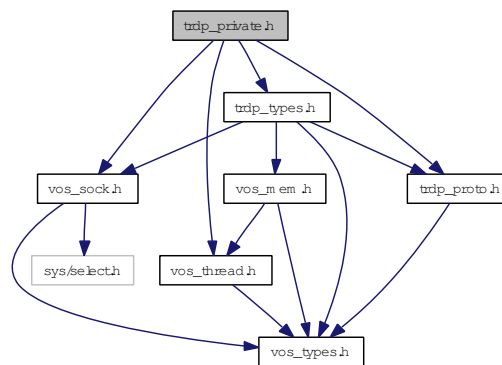


5.15 trdp_private.h File Reference

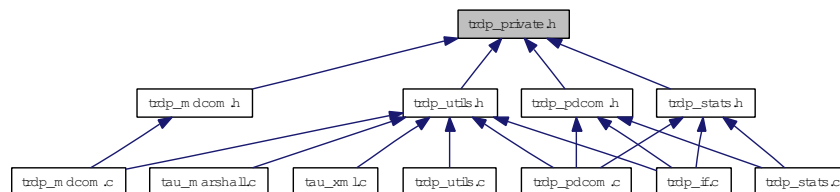
Typedefs for TRDP communication.

```
#include "trdp_types.h"
#include "trdp_proto.h"
#include "vos_thread.h"
#include "vos_sock.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_SOCKET_TCP](#)
TCP parameters.
- struct [TRDP_SOCKETS](#)
Socket item.
- struct [GNU_PACKED](#)
TRDP process data header - network order and alignment.
- struct [GNU_PACKED](#)
TRDP process data header - network order and alignment.

- struct [PD_ELE](#)
Queue element for PD packets to send or receive.
- struct [MD_LIS_ELE](#)
Queue element for MD listeners (UDP and TCP).
- struct [TRDP_MD_TCP](#)
Tcp connection parameters.
- struct [MD_ELE](#)
Session queue element for MD (UDP and TCP).
- struct [TRDP_TCP_FD_T](#)
TCP file descriptor parameters.
- struct [TRDP_SESSION](#)
Session/application variables store.

Defines

- #define [TRDP_TIMER_GRANULARITY](#) 10000
granularity in us
- #define [TRDP_TIMER_FOREVER](#) 0xffffffff
granularity in us
- #define [TRDP_MD_DEFAULT_REPLY_TIMEOUT](#) 5000000
default reply time out 5s
- #define [TRDP_MD_DEFAULT_CONFIRM_TIMEOUT](#) 1000000
default confirm time out 1s
- #define [TRDP_MD_DEFAULT_CONNECTION_TIMEOUT](#) 60000000
Socket connection time out 1 minute.
- #define [TRDP_MD_DEFAULT_SENDING_TIMEOUT](#) 5000000
Socket sending time out 5s.
- #define [TRDP_PROCESS_DEFAULT_CYCLE_TIME](#) 10000
Default cycle time for TRDP process.
- #define [TRDP_PROCESS_DEFAULT_PRIORITY](#) 64
Default priority of TRDP process.
- #define [TRDP_PROCESS_DEFAULT_OPTIONS](#) TRDP_OPTION_TRAFFIC_SHAPING
Default options for TRDP process.
- #define [TRDP_DEBUG_DEFAULT_FILE_SIZE](#) 65536
Default maximum size of log file.

Typedefs

- typedef struct [TRDP_HANDLE](#) [TRDP_ADDRESSES_T](#)
Hidden handle definition, used as unique addressing item.
- typedef struct [TRDP_SOCKET_TCP](#) [TRDP_SOCKET_TCP_T](#)
TCP parameters.
- 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_LIS_ELE](#) [MD_LIS_ELE_T](#)
Queue element for MD listeners (UDP and TCP).
- typedef struct [TRDP_MD_TCP](#) [TRDP_MD_TCP_T](#)
Tcp connection parameters.
- typedef struct [MD_ELE](#) [MD_ELE_T](#)
Session queue element for MD (UDP and TCP).
- typedef struct [TRDP_SESSION](#) [TRDP_SESSION_T](#)
Session/application variables store.

Enumerations

- enum [TRDP_MD_ELE_ST_T](#) {
[TRDP_ST_NONE](#) = 0,
[TRDP_ST_TX_NOTIFY_ARM](#) = 1,
[TRDP_ST_TX_REQUEST_ARM](#) = 2,
[TRDP_ST_TX_REPLY_ARM](#) = 3,
[TRDP_ST_TX_REPLYQUERY_ARM](#) = 4,
[TRDP_ST_TX_CONFIRM_ARM](#) = 5,
[TRDP_ST_RX_READY](#) = 6,
[TRDP_ST_TX_REQUEST_W4REPLY](#) = 7,
[TRDP_ST_RX_REPLYQUERY_W4C](#) = 8,
[TRDP_ST_RX_REQ_W4AP_REPLY](#) = 9,
[TRDP_ST_TX_REQ_W4AP_CONFIRM](#) = 10,
[TRDP_ST_RX_REPLY_SENT](#) = 11,
[TRDP_ST_RX_NOTIFY_RECEIVED](#) = 12,
[TRDP_ST_TX_REPLY_RECEIVED](#) = 13,
[TRDP_ST_RX_CONF_RECEIVED](#) = 14 }
Internal MD state.

- enum [TRDP_PRIV_FLAGS_T](#) { ,
[TRDP_TIMED_OUT](#) = 0x2,
[TRDP_INVALID_DATA](#) = 0x4,
[TRDP_REQ_2B_SENT](#) = 0x8,
[TRDP_PULL_SUB](#) = 0x10,
[TRDP_REDUNDANT](#) = 0x20 }

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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_private.h](#) 950 2013-06-13 13:51:41Z 97025

5.15.2 Enumeration Type Documentation

5.15.2.1 enum TRDP_MD_ELE_ST_T

Internal MD state.

Enumerator:

TRDP_ST_NONE neutral value
TRDP_ST_TX_NOTIFY_ARM ready to send notify MD
TRDP_ST_TX_REQUEST_ARM ready to send request MD
TRDP_ST_TX_REPLY_ARM ready to send reply MD

TRDP_ST_TX_REPLYQUERY_ARM ready to send reply with confirm request MD
TRDP_ST_TX_CONFIRM_ARM ready to send confirm MD
TRDP_ST_RX_READY armed listener
TRDP_ST_TX_REQUEST_W4REPLY request sent, wait for reply
TRDP_ST_RX_REPLYQUERY_W4C reply send, with confirm request MD
TRDP_ST_RX_REQ_W4AP_REPLY request received, wait for application reply send
TRDP_ST_TX_REQ_W4AP_CONFIRM reply conf.
 rq. tx, wait for application conf send
TRDP_ST_RX_REPLY_SENT reply sent
TRDP_ST_RX_NOTIFY_RECEIVED notification received, wait for application to accept
TRDP_ST_TX_REPLY_RECEIVED reply received
TRDP_ST_RX_CONF_RECEIVED confirmation received

5.15.2.2 enum TRDP_PRIV_FLAGS_T

Internal flags for packets.

Enumerator:

TRDP_TIMED_OUT if set, inform the user
TRDP_INVALID_DATA if set, inform the user
TRDP_REQ_2B_SENT if set, the request needs to be sent
TRDP_PULL_SUB if set, its a PULL subscription
TRDP_REDUNDANT if set, packet should not be sent (redundant)

5.15.2.3 enum TRDP SOCK_TYPE_T

Socket usage.

Enumerator:

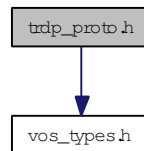
TRDP SOCK_PD Socket is used for UDP process data.
TRDP SOCK_MD_UDP Socket is used for UDP message data.
TRDP SOCK_MD_TCP Socket is used for TCP message data.

5.16 trdp_proto.h File Reference

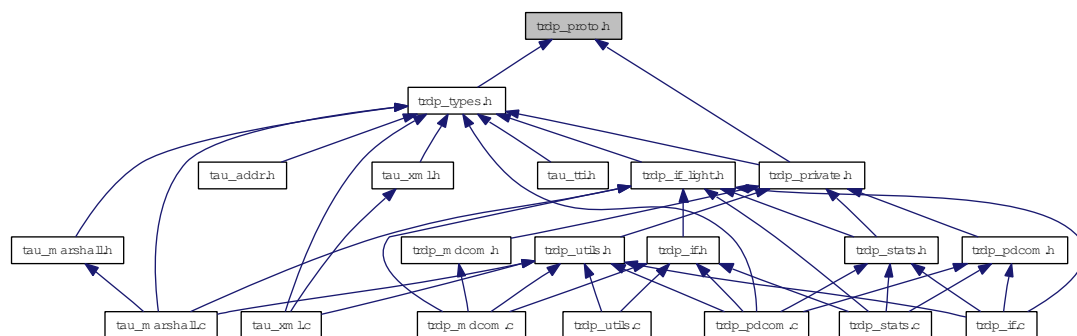
Definitions for the TRDP protocol.

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

Include dependency graph for trdp_proto.h:



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



Data Structures

- struct [GNU_PACKED](#)
TRDP process data header - network order and alignment.
- struct [GNU_PACKED](#)
TRDP process data header - network order and alignment.

Defines

- #define [TRDP_PD_UDP_PORT](#) 20548
process data UDP port
- #define [TRDP_MD_UDP_PORT](#) 20550
message data UDP port
- #define [TRDP_MD_TCP_PORT](#) 20550
message data TCP port
- #define [TRDP_PROTO_VER](#) 0x0100

Protocol version.

- #define [TRDP_PROTOCOL_VERSION_CHECK_MASK](#) 0xFF00
Version check, two digits are relevant.
- #define [TRDP_SESS_ID_SIZE](#) 16
Session ID (UUID) size in MD header.
- #define [TRDP_DEST_URI_SIZE](#) 32
max.
- #define [TRDP_MIN_PD_HEADER_SIZE](#) sizeof(PD_HEADER_T)
PD header size with FCS.
- #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 [TRDP_VAR_SIZE](#) 0
Variable size dataset.
- #define [TRDP_COMID_ECHO](#) 10
TRDP reserved COMIDs in the range 1 .
- #define [TRDP_STATISTICS_REQUEST_DSID](#) 31
TRDP reserved data set ids in the range 1 .

Enumerations

- enum [TRDP_MSG_T](#) {
[TRDP_MSG_PD](#) = 0x5064,
[TRDP_MSG_PP](#) = 0x5070,
[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 }
```

Message Types.

5.16.1 Detailed Description

Definitions for the TRDP protocol.

TRDP internal type definitions

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2013.

Id

[trdp_proto.h](#) 795 2013-05-10 10:12:58Z bloehr

5.16.2 Define Documentation

5.16.2.1 `#define TRDP_COMID_ECHO 10`

TRDP reserved COMIDs in the range 1 .

.. 1000

5.16.2.2 `#define TRDP_DEST_URI_SIZE 32`

max.

Dest URI size in MD header

5.16.2.3 `#define TRDP_MAX_FILE_NAME_LEN 128`

path and file name length incl.

terminating '0'

5.16.2.4 #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: $7 + (6 * 15) + 5 * (\text{sizeof}(\text{separator})) + 1(\text{terminating } 0)$ to facilitate alignment the size will be increased by 1 byte label length incl. terminating '0'

5.16.2.5 #define TRDP_MAX_URI_HOST_LEN (4 * TRDP_MAX_LABEL_LEN)

URI host part length incl.

terminating '0'

5.16.2.6 #define TRDP_MAX_URI_LEN ((6 * TRDP_MAX_LABEL_LEN) + 8)

URI length incl.

terminating '0' and 1 padding byte

5.16.2.7 #define TRDP_MAX_URI_USER_LEN (2 * TRDP_MAX_LABEL_LEN)

URI user part incl.

terminating '0'

5.16.2.8 #define TRDP_STATISTICS_REQUEST_DSID 31

TRDP reserved data set ids in the range 1 .

.. 1000

5.16.3 Enumeration Type Documentation

5.16.3.1 enum TRDP_MSG_T

Message Types.

Enumerator:

```

TRDP_MSG_PD  'Pd' PD Data
TRDP_MSG_PP  'Pp' PD Data (Pull 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

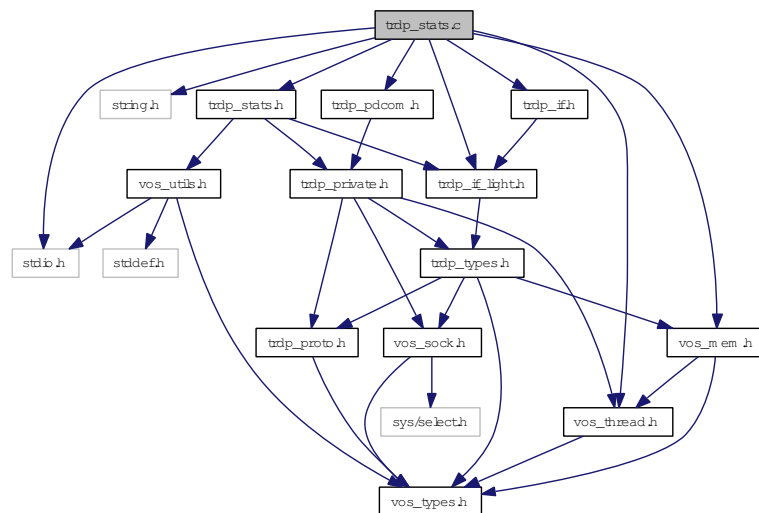
```

5.17 trdp_stats.c File Reference

Statistics functions for TRDP communication.

```
#include <stdio.h>
#include <string.h>
#include "trdp_stats.h"
#include "trdp_if_light.h"
#include "trdp_if.h"
#include "trdp_pdcom.h"
#include "vos_mem.h"
#include "vos_thread.h"
```

Include dependency graph for trdp_stats.c:



Functions

- void [trdp_UpdateStats](#) (TRDP_APP_SESSION_T appHandle)
Update the statistics.
- void [trdp_initStats](#) (TRDP_APP_SESSION_T appHandle)
Init statistics.
- EXT_DECL [TRDP_ERR_T tlc_resetStatistics](#) (TRDP_APP_SESSION_T appHandle)
Reset statistics.
- EXT_DECL [TRDP_ERR_T tlc_getStatistics](#) (TRDP_APP_SESSION_T appHandle, [TRDP_STATISTICS_T](#) *pStatistics)
Return statistics.
- EXT_DECL [TRDP_ERR_T tlc_getSubsStatistics](#) (TRDP_APP_SESSION_T appHandle, UINT16 *pNumSubs, [TRDP_SUBS_STATISTICS_T](#) *pStatistics)

Return PD subscription statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getPubStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumPub, [TRDP_PUB_STATISTICS_T](#) *pStatistics)

Return PD publish statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getListStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumList, [TRDP_LIST_STATISTICS_T](#) *pStatistics)

Return MD listener statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getRedStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumRed, [TRDP_RED_STATISTICS_T](#) *pStatistics)

Return redundancy group statistics.

- EXT_DECL [TRDP_ERR_T](#) [tlc_getJoinStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumJoin, UINT32 *pIpAddr)

Return join statistics.

- void [trdp_pdPrepareStats](#) ([TRDP_APP_SESSION_T](#) appHandle, [PD_ELE_T](#) *pPacket)

Fill the statistics packet.

5.17.1 Detailed Description

Statistics functions for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_stats.c](#) 950 2013-06-13 13:51:41Z 97025

5.17.2 Function Documentation

5.17.2.1 EXT_DECL [TRDP_ERR_T](#) [tlc_getJoinStatistics](#) ([TRDP_APP_SESSION_T](#) appHandle, UINT16 *pNumJoin, UINT32 *pIpAddr)

Return join statistics.

Memory for statistics information must be provided by the user.

Parameters:

← **appHandle** the handle returned by [tlc_openSession](#)

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

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

Return values:

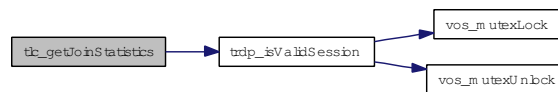
TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR there are more items than requested

Here is the call graph for this function:



5.17.2.2 EXT_DECL TRDP_ERR_T tlc_getListStatistics (TRDP_APP_SESSION_T appHandle, UINT16 * pNumList, TRDP_LIST_STATISTICS_T * pStatistics)

Return MD listener statistics.

Memory for statistics information must be provided by the user.

Parameters:

← *appHandle* the handle returned by tlc_openSession

↔ *pNumList* Pointer to the number of listeners

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

Return values:

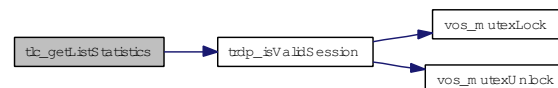
TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR there are more subscriptions than requested

Here is the call graph for this function:



5.17.2.3 EXT_DECL TRDP_ERR_T tlc_getPubStatistics (TRDP_APP_SESSION_T appHandle, UINT16 * pNumPub, TRDP_PUB_STATISTICS_T * pStatistics)

Return PD publish statistics.

Memory for statistics information must be provided by the user.

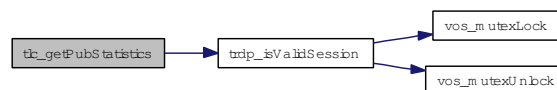
Parameters:

- ← **appHandle** the handle returned by `tlc_openSession`
- ↔ **pNumPub** Pointer to the number of publishers
- **pStatistics** 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
- TRDP_MEM_ERR** there are more subscriptions than requested

Here is the call graph for this function:



5.17.2.4 EXT_DECL TRDP_ERR_T tlc_getRedStatistics (TRDP_APP_SESSION_T appHandle, UINT16 * pNumRed, TRDP_RED_STATISTICS_T * pStatistics)

Return redundancy group statistics.

Memory for statistics information must be provided by the user.

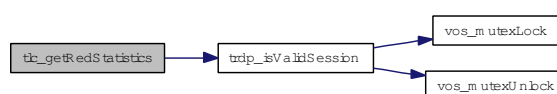
Parameters:

- ← **appHandle** the handle returned by `tlc_openSession`
- ↔ **pNumRed** Pointer to the number of redundancy groups
- **pStatistics** 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
- TRDP_MEM_ERR** there are more subscriptions than requested

Here is the call graph for this function:



5.17.2.5 EXT_DECL TRDP_ERR_T tlc_getStatistics (TRDP_APP_SESSION_T *appHandle*, TRDP_STATISTICS_T * *pStatistics*)

Return statistics.

Memory for statistics information must be provided by the user.

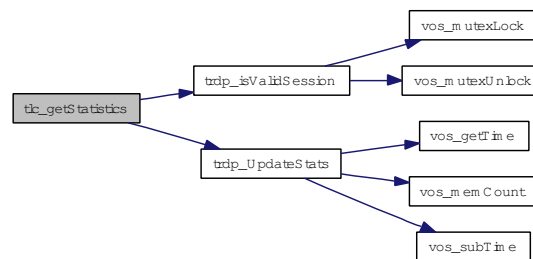
Parameters:

- ← *appHandle* the handle returned by tlc_openSession
- *pStatistics* Pointer to 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:



5.17.2.6 EXT_DECL TRDP_ERR_T tlc_getSubsStatistics (TRDP_APP_SESSION_T *appHandle*, UINT16 * *pNumSubs*, TRDP_SUBS_STATISTICS_T * *pStatistics*)

Return PD subscription statistics.

Memory for statistics information must be provided by the user.

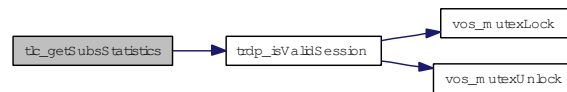
Parameters:

- ← *appHandle* the handle returned by tlc_openSession
- ↔ *pNumSubs* In: The number of subscriptions requested Out: Number of subscriptions returned
- ↔ *pStatistics* Pointer to an array with the subscription statistics information

Return values:

- TRDP_NO_ERR** no error
- TRDP_NOINIT_ERR** handle invalid
- TRDP_PARAM_ERR** parameter error
- TRDP_MEM_ERR** there are more subscriptions than requested

Here is the call graph for this function:



5.17.2.7 EXT_DECL TRDP_ERR_T tlc_resetStatistics (TRDP_APP_SESSION_T *appHandle*)

Reset statistics.

Parameters:

← *appHandle* the handle returned by tlc_openSession

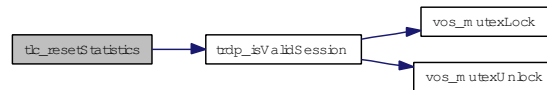
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.17.2.8 void trdp_initStats (TRDP_APP_SESSION_T *appHandle*)

Init statistics.

Clear the stats structure for a session.

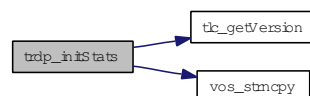
Parameters:

← *appHandle* the handle returned by tlc_openSession

< host name

< leader host name

Here is the call graph for this function:



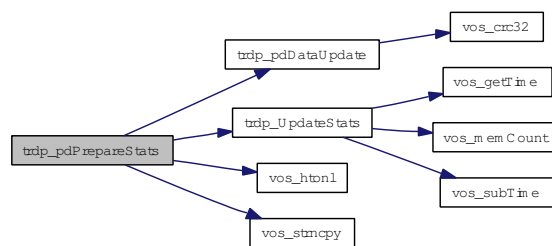
5.17.2.9 void trdp_pdPrepareStats (TRDP_APP_SESSION_T *appHandle*, PD_ELE_T * *pPacket*)

Fill the statistics packet.

Parameters:

- ← ***appHandle*** the handle returned by `tlc_openSession`
- ↔ ***pPacket*** pointer to the packet to fill

Here is the call graph for this function:



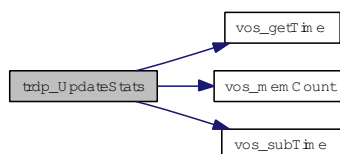
5.17.2.10 void trdp_UpdateStats (TRDP_APP_SESSION_T *appHandle*)

Update the statistics.

Parameters:

- ← ***appHandle*** the handle returned by `tlc_openSession`

Here is the call graph for this function:

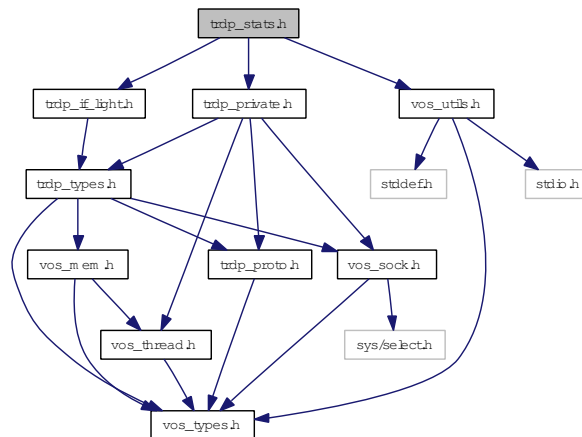


5.18 trdp_stats.h File Reference

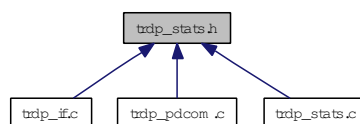
Statistics for TRDP communication.

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



Functions

- void [trdp_initStats](#) (TRDP_APP_SESSION_T appHandle)
Init statistics.
- void [trdp_pdPrepareStats](#) (TRDP_APP_SESSION_T appHandle, PD_ELE_T *pPacket)
Fill the statistics packet.

5.18.1 Detailed Description

Statistics for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_stats.h](#) 950 2013-06-13 13:51:41Z 97025

5.18.2 Function Documentation**5.18.2.1 void trdp_initStats (TRDP_APP_SESSION_T *appHandle*)**

Init statistics.

Clear the stats structure for a session.

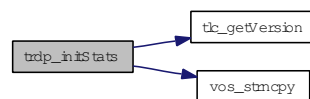
Parameters:

← *appHandle* the handle returned by `tlc_openSession`

< host name

< leader host name

Here is the call graph for this function:

**5.18.2.2 void trdp_pdPrepareStats (TRDP_APP_SESSION_T *appHandle*, PD_ELE_T **pPacket*)**

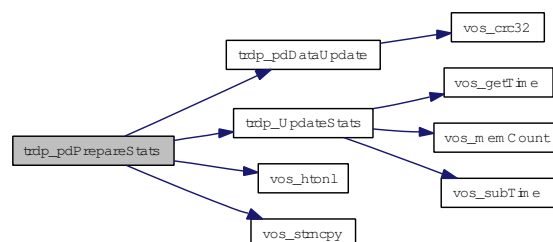
Fill the statistics packet.

Parameters:

← *appHandle* the handle returned by `tlc_openSession`

↔ *pPacket* pointer to the packet to fill

Here is the call graph for this function:



- struct [TRDP_DATASET](#)
Dataset definition.
- struct [TRDP_COMID_DSID_MAP_T](#)
ComId - data set mapping element 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](#)
Enumeration type for memory pre-fragmentation, reuse of VOS definition.
- struct [TRDP_PROCESS_CONFIG_T](#)
Various flags/general TRDP options for library initialization.

Defines

- #define [USE_HEAP](#) 0
If this is set, we can allocate dynamically memory.

Typedefs

- typedef VOS_IP4_ADDR_T [TRDP_IP_ADDR_T](#)
TRDP general type definitions.
- typedef [VOS_TIME_T](#) [TRDP_TIME_T](#)
Timer value compatible with timeval / select.
- typedef [VOS_FDS_T](#) [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 struct [TRDP_DATASET](#) [TRDP_DATASET_T](#)
Dataset definition.
- typedef [TRDP_DATASET_T](#) * [pTRDP_DATASET_T](#)
Array of pointers to dataset.
- 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, UINT8 *pSrc, UINT8 *pDst, UINT32 *pDstSize, [TRDP_DATASET_T](#) **ppCachedDS)
Function type for marshalling .
- typedef [TRDP_ERR_T](#)(* [TRDP_UNMARSHALL_T](#))(void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDst, UINT32 *pDstSize, [TRDP_DATASET_T](#) **ppCachedDS)
Function type for unmarshalling.
- typedef void(* [TRDP_PD_CALLBACK_T](#))(void *pRefCon, [TRDP_APP_SESSION_T](#) appHandle, 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, [TRDP_APP_SESSION_T](#) appHandle, const [TRDP_MD_INFO_T](#) *pMsg, UINT8 *pData, UINT32 dataSize)
Callback for receiving indications, timeouts, releases, responses.

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_THREAD_ERR = -13,
TRDP_BLOCK_ERR = -14,
TRDP_INTEGRATION_ERR = -15,
TRDP_NOCONN_ERR = -16,
TRDP_NOSESSION_ERR = -30,
TRDP_SESSION_ABORT_ERR = -31,
TRDP_NOSUB_ERR = -32,
TRDP_NOPUB_ERR = -33,
TRDP_NOLIST_ERR = -34,
TRDP_CRC_ERR = -35,
TRDP_WIRE_ERR = -36,
TRDP_TOPO_ERR = -37,
TRDP_COMID_ERR = -38,
TRDP_STATE_ERR = -39,
TRDP_APP_TIMEOUT_ERR = -40,
TRDP_APP_REPLYTO_ERR = -41,
TRDP_APP_CONFIRMTO_ERR = -42,
TRDP_REPLYTO_ERR = -43,
TRDP_CONFIRMTO_ERR = -44,
TRDP_REQCONFIRMTO_ERR = -45,
TRDP_PACKET_ERR = -46,
TRDP_UNKNOWN_ERR = -99 }

```

Return codes for all API functions, -1.

- enum `TRDP_REPLY_STATUS_T`
TRDP data transfer type definitions.
- enum `TRDP_FLAGS_T` {
`TRDP_FLAGS_DEFAULT` = 0,
`TRDP_FLAGS_NONE` = 0x01,
`TRDP_FLAGS_MARSHALL` = 0x02,
`TRDP_FLAGS_CALLBACK` = 0x04,
`TRDP_FLAGS_TCP` = 0x08 }

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` {
`TRDP_TO_DEFAULT` = 0,
`TRDP_TO_SET_TO_ZERO` = 1,
`TRDP_TO_KEEP_LAST_VALUE` = 2 }

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_TIMEDATE32` = 14,
`TRDP_TIMEDATE48` = 15,
`TRDP_TIMEDATE64` = 16,
`TRDP_TYPE_MAX` = 30 }

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.

5.19.1 Detailed Description

Typedefs for TRDP communication.

F

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_types.h](#) 920 2013-06-10 15:40:33Z aweiss

5.19.2 Typedef Documentation

5.19.2.1 typedef VOS_IP4_ADDR_T TRDP_IP_ADDR_T

TRDP general type definitions.

5.19.2.2 typedef TRDP_ERR_T(* TRDP_MARSHALL_T)(void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDst, UINT32 *pDstSize, TRDP_DATASET_T **ppCachedDS)

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
- ↔ **ppCachedDS* pointer to pointer of cached dataset

Return values:

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

5.19.2.3 `typedef void(* TRDP_MD_CALLBACK_T)(void *pRefCon, TRDP_APP_SESSION_T appHandle, const TRDP_MD_INFO_T *pMsg, UINT8 *pData, UINT32 dataSize)`

Callback for receiving indications, timeouts, releases, responses.

Parameters:

- ← *appHandle* handle returned also by `tlc_init`
- ← *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 !!!!

5.19.2.4 `typedef void(* TRDP_PD_CALLBACK_T)(void *pRefCon, TRDP_APP_SESSION_T appHandle, const TRDP_PD_INFO_T *pMsg, UINT8 *pData, UINT32 dataSize)`

Callback for receiving indications, timeouts, releases, responses.

Parameters:

- ← *pRefCon* pointer to user context
- ← *appHandle* application handle returned by `tlc_openSession`
- ← *pMsg* pointer to received message information
- ← *pData* pointer to received data
- ← *dataSize* size of received data pointer to received data excl. padding and FCS !!!!

5.19.2.5 `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.

5.19.2.6 `typedef VOS_TIME_T TRDP_TIME_T`

Timer value compatible with `timeval` / `select`.

Relative or absolute date, depending on usage

5.19.2.7 `typedef TRDP_ERR_T(* TRDP_UNMARSHALL_T)(void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDst, UINT32 *pDstSize, TRDP_DATASET_T **ppCachedDS)`

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
 ↔ **ppCachedDS* pointer to pointer of cached dataset

Return values:

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

5.19.3 Enumeration Type Documentation**5.19.3.1 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_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 microseconds (== struct timeval)
TRDP_TYPE_MAX Values greater are considered nested datasets.

5.19.3.2 enum TRDP_ERR_T

Return codes for all API functions, -1.

.-29 taken over from vos

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 Timeout.
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_THREAD_ERR Thread error.
TRDP_BLOCK_ERR System call would have blocked in blocking mode.
TRDP_INTEGRATION_ERR Alignment or endianness for selected target wrong.
TRDP_NOCONN_ERR No TCP connection.
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_WIRE_ERR Wire.
TRDP_TOPO_ERR Invalid topo count.
TRDP_COMID_ERR Unknown ComId.
TRDP_STATE_ERR Call in wrong state.
TRDP_APP_TIMEOUT_ERR Application Timeout.
TRDP_APP_REPLYTO_ERR Application Reply Sent Timeout.
TRDP_APP_CONFIRMTO_ERR Application Confirm Sent Timeout.
TRDP_REPLYTO_ERR Protocol Reply Timeout.
TRDP_CONFIRMTO_ERR Protocol Confirm Timeout.
TRDP_REQCONFIRMTO_ERR Protocol Confirm Timeout (Request sender).
TRDP_PACKET_ERR Incomplete message data packet.
TRDP_UNKNOWN_ERR Unspecified error.

5.19.3.3 enum TRDP_FLAGS_T

Various flags for PD and MD packets.

Enumerator:

TRDP_FLAGS_DEFAULT Default value defined in tlc_openDession will be taken.
TRDP_FLAGS_NONE No flags set.
TRDP_FLAGS_MARSHALL Optional marshalling/unmarshalling in TRDP stack.
TRDP_FLAGS_CALLBACK Use of callback function.
TRDP_FLAGS_TCP Use TCP for message data.

5.19.3.4 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.

5.19.3.5 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.

5.19.3.6 enum TRDP_REPLY_STATUS_T

TRDP data transfer type definitions.

Reply status messages

5.19.3.7 enum TRDP_TO_BEHAVIOR_T

How invalid PD shall be handled.

Enumerator:

TRDP_TO_DEFAULT Default value defined in tlc_openDession will be taken.

TRDP_TO_SET_TO_ZERO If set, data will be reset to zero on time out.

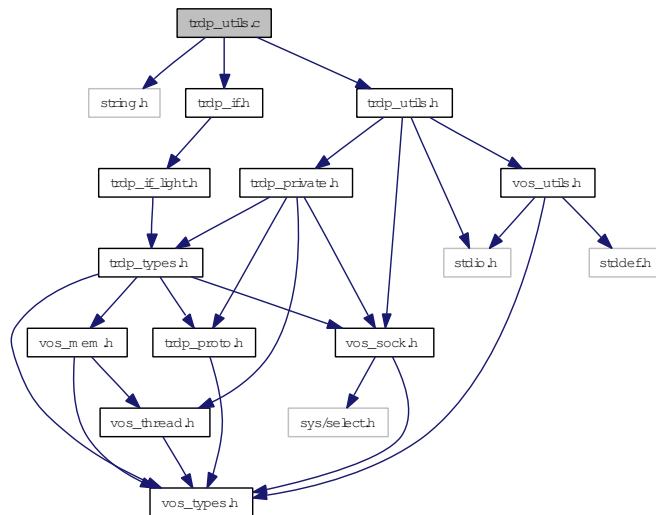
TRDP_TO_KEEP_LAST_VALUE If set, last received values will be returned.

5.20 trdp_utils.c File Reference

Helper functions for TRDP communication.

```
#include <string.h>
#include "trdp_if.h"
#include "trdp_utils.h"
```

Include dependency graph for trdp_utils.c:



Functions

- **BOOL** `trdp_SockIsJoined` (const [TRDP_IP_ADDR_T](#) mcList[VOS_MAX_MULTICAST_CNT], [TRDP_IP_ADDR_T](#) mcGroup)
Check if a mc group is in the list.
- **BOOL** `trdp_SockAddJoin` ([TRDP_IP_ADDR_T](#) mcList[VOS_MAX_MULTICAST_CNT], [TRDP_IP_ADDR_T](#) mcGroup)
Add mc group to the list.
- **BOOL** `trdp_SockDelJoin` ([TRDP_IP_ADDR_T](#) mcList[VOS_MAX_MULTICAST_CNT], [TRDP_IP_ADDR_T](#) mcGroup)
remove mc group from the list
- **int** `am_big_endian` ()
Determine if we are Big or Little endian.
- **UINT32** `trdp_packetSizePD` (UINT32 dataSize)
Get the packet size from the raw data size.
- **UINT32** `trdp_packetSizeMD` (UINT32 dataSize)
Get the packet size from the raw data size.

- `PD_ELE_T * trdp_queueFindComId (PD_ELE_T *pHead, UINT32 comId)`
Return the element with same comId.
- `PD_ELE_T * trdp_queueFindPubAddr (PD_ELE_T *pHead, TRDP_ADDRESSES_T *addr)`
Return the element with same comId and IP addresses.
- `PD_ELE_T * trdp_queueFindSubAddr (PD_ELE_T *pHead, TRDP_ADDRESSES_T *addr)`
Return the element with same comId and IP addresses.
- `MD_ELE_T * trdp_MDqueueFindAddr (MD_ELE_T *pHead, TRDP_ADDRESSES_T *addr)`
Return the element with same comId from MD queue.
- `void trdp_queueDelElement (PD_ELE_T **ppHead, PD_ELE_T *pDelete)`
Delete an element.
- `void trdp_MDqueueDelElement (MD_ELE_T **ppHead, MD_ELE_T *pDelete)`
Delete an element from MD queue.
- `void trdp_queueAppLast (PD_ELE_T **ppHead, PD_ELE_T *pNew)`
Append an element at end of queue.
- `void trdp_MDqueueAppLast (MD_ELE_T **ppHead, MD_ELE_T *pNew)`
Append an element at end of queue.
- `void trdp_queueInsFirst (PD_ELE_T **ppHead, PD_ELE_T *pNew)`
Insert an element at front of queue.
- `void trdp_MDqueueInsFirst (MD_ELE_T **ppHead, MD_ELE_T *pNew)`
Insert an element at front of MD queue.
- `void trdp_initSockets (TRDP_SOCKETS_T iface[])`
Handle the socket pool: Initialize it.
- `TRDP_ERR_T trdp_requestSocket (TRDP_SOCKETS_T iface[], UINT32 port, const TRDP_SEND_PARAM_T *params, TRDP_IP_ADDR_T srcIP, TRDP_IP_ADDR_T mcGroup, TRDP_SOCK_TYPE_T usage, TRDP_OPTION_T options, BOOL rcvMostly, INT32 useSocket, INT32 *pIndex, TRDP_IP_ADDR_T cornerIp)`
Handle the socket pool: Request a socket from our socket pool First we loop through the socket pool and check if there is already a socket which would suit us.
- `void trdp_releaseSocket (TRDP_SOCKETS_T iface[], INT32 lIndex, UINT32 connectTimeout, BOOL checkAll)`
Handle the socket pool: if a received TCP socket is unused, the socket connection timeout is started.
- `UINT32 trdp_getSeqCnt (UINT32 comId, TRDP_MSG_T msgType, TRDP_IP_ADDR_T srcIPAddr)`
Get the initial sequence counter for the comId/message type and subnet (source IP).
- `BOOL trdp_isRcvSeqCnt (UINT32 seqCnt, UINT32 comId, TRDP_MSG_T msgType, TRDP_IP_ADDR_T srcIP)`

Check if the sequence counter for the comID/message type and subnet (source IP) has already been received.

- **BOOL** `trdp_isAddressed` (const `TRDP_URI_USER_T` listUri, const `TRDP_URI_USER_T` destUri)

Check if listener URI is in addressing range of destination URI.

5.20.1 Detailed Description

Helper functions for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2013.

Id

[trdp_utils.c](#) 962 2013-06-14 08:06:30Z bloehr

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 `UINT32 trdp_getSeqCnt (UINT32 comId, TRDP_MSG_T msgType, TRDP_IP_ADDR_T srcIpAddr)`

Get the initial sequence counter for the comID/message type and subnet (source IP).

If the comID/srcIP is not found elsewhere, return 0 - else return its current sequence number (the redundant packet needs the same seqNo)

Note: The standard demands that sequenceCounter is managed per comID/msgType at each publisher, but shall be the same for redundant telegrams (subnet/srcIP).

Parameters:

← *comId* comID to look for

← *msgType* PD/MD type

← *srcIpAddr* Source IP address

Return values:

return the sequence number

Here is the call graph for this function:



5.20.2.3 void trdp_initSockets (TRDP_SOCKETS_T *iface*[])

Handle the socket pool: Initialize it.

Parameters:

← *iface* pointer to the socket pool

5.20.2.4 BOOL trdp_isAddressed (const TRDP_URI_USER_T *listUri*, const TRDP_URI_USER_T *destUri*)

Check if listener URI is in addressing range of destination URI.

Parameters:

← *listUri* Null terminated listener URI string to compare

← *destUri* Null terminated destination URI string to compare

Return values:

FALSE - not in addressing range

TRUE - listener URI is in addressing range of destination URI

Here is the call graph for this function:



5.20.2.5 BOOL trdp_isRcvSeqCnt (UINT32 *seqCnt*, UINT32 *comId*, TRDP_MSG_T *msgType*, TRDP_IP_ADDR_T *srcIP*)

Check if the sequence counter for the comID/message type and subnet (source IP) has already been received.

Note: The standard demands that sequenceCounter is managed per comID/msgType at each publisher, but shall be the same for redundant telegrams (subnet/srcIP).

Parameters:

- ← *seqCnt* sequence counter received
- ← *comId* comID to look for
- ← *msgType* PD/MD type
- ← *srcIP* Source IP address

Return values:

return the sequence number

Here is the call graph for this function:



5.20.2.6 void trdp_MDqueueAppLast (MD_ELE_T ** *ppHead*, MD_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

5.20.2.7 void trdp_MDqueueDelElement (MD_ELE_T ** *ppHead*, MD_ELE_T * *pDelete*)

Delete an element from MD queue.

Parameters:

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

5.20.2.8 MD_ELE_T* trdp_MDqueueFindAddr (MD_ELE_T * *pHead*, TRDP_ADDRESSES_T * *addr*)

Return the element with same comId from MD queue.

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

5.20.2.9 void trdp_MDqueueInsFirst (MD_ELE_T ** *ppHead*, MD_ELE_T * *pNew*)

Insert an element at front of MD queue.

Parameters:

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

5.20.2.10 UINT32 trdp_packetSizeMD (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

5.20.2.11 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

5.20.2.12 void trdp_queueAppLast (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

5.20.2.13 void trdp_queueDelElement (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

5.20.2.14 PD_ELE_T* trdp_queueFindComId (PD_ELE_T * *pHead*, UINT32 *comId*)

Return the element with same comId.

Parameters:

- ← *pHead* pointer to head of queue
- ← *comId* ComID to search for

Return values:

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

5.20.2.15 PD_ELE_T* trdp_queueFindPubAddr (PD_ELE_T * *pHead*, TRDP_ADDRESSES_T * *addr*)

Return the element with same comId and IP addresses.

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

5.20.2.16 PD_ELE_T* trdp_queueFindSubAddr (PD_ELE_T * *pHead*, TRDP_ADDRESSES_T * *addr*)

Return the element with same comId and IP addresses.

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

5.20.2.17 void trdp_queueInsFirst (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

5.20.2.18 void trdp_releaseSocket (TRDP_SOCKETS_T *iface*[], INT32 *lIndex*, UINT32 *connectTimeout*, BOOL *checkAll*)

Handle the socket pool: if a received TCP socket is unused, the socket connection timeout is started.

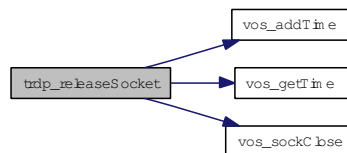
Handle the socket pool: Release a socket from our socket pool.

In Udp, Release a socket from our socket pool

Parameters:

- ↔ *iface* socket pool
- ← *lIndex* index of socket to release
- ← *connectTimeout* time out
- ← *checkAll* release all TCP pending sockets

Here is the call graph for this function:



5.20.2.19 TRDP_ERR_T trdp_requestSocket (TRDP_SOCKETS_T *iface*[], UINT32 *port*, const TRDP_SEND_PARAM_T **params*, TRDP_IP_ADDR_T *srcIP*, TRDP_IP_ADDR_T *mcGroup*, TRDP SOCK_TYPE_T *usage*, TRDP_OPTION_T *options*, BOOL *rcvMostly*, INT32 *useSocket*, INT32 **pIndex*, TRDP_IP_ADDR_T *cornerIp*)

Handle the socket pool: Request a socket from our socket pool First we loop through the socket pool and check if there is already a socket which would suit us.

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

If a multicast group should be joined, we do that on an otherwise suitable socket - up to 20 multicast groups can be joined per socket. If a socket for multicast publishing is requested, we also use the source IP to determine the interface for outgoing multicast traffic.

Parameters:

- ↔ *iface* socket pool
- ← *port* port to use
- ← *params* parameters to use
- ← *srcIP* IP to bind to (0 = any address)
- ← *mcGroup* MC group to join (0 = do not join)
- ← *usage* type and port to bind to (PD, MD/UDP, MD/TCP)
- ← *options* blocking/nonblocking
- ← *rcvMostly* primarily used for receiving (tbd: bind on sender, too?)
- *useSocket* socket to use, do not open a new one
- *pIndex* returned index of socket pool

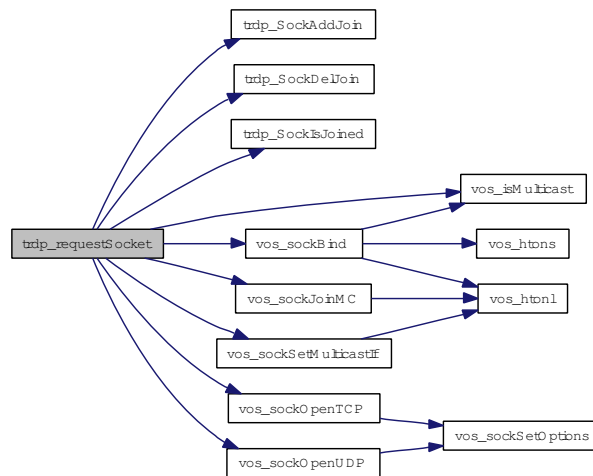
← *cornerIp* only used for receiving

Return values:

TRDP_NO_ERR

TRDP_PARAM_ERR

Here is the call graph for this function:



5.20.2.20 **BOOL trdp_SockAddJoin (TRDP_IP_ADDR_T mcList[VOS_MAX_MULTICAST_CNT], TRDP_IP_ADDR_T mcGroup)**

Add mc group to the list.

Parameters:

← *mcList[]* List of multicast groups

← *mcGroup* multicast group

Return values:

1 if added 0 if list is full

5.20.2.21 **BOOL trdp_SockDelJoin (TRDP_IP_ADDR_T mcList[VOS_MAX_MULTICAST_CNT], TRDP_IP_ADDR_T mcGroup)**

remove mc group from the list

Parameters:

← *mcList[]* List of multicast groups

← *mcGroup* multicast group

Return values:

1 if deleted 0 was not in list

5.20.2.22 **BOOL** trdp_SockIsJoined (const TRDP_IP_ADDR_T *mcList*[VOS_MAX_MULTICAST_CNT], TRDP_IP_ADDR_T *mcGroup*)

Check if a mc group is in the list.

Parameters:

← *mcList*[] List of multicast groups

← *mcGroup* multicast group

Return values:

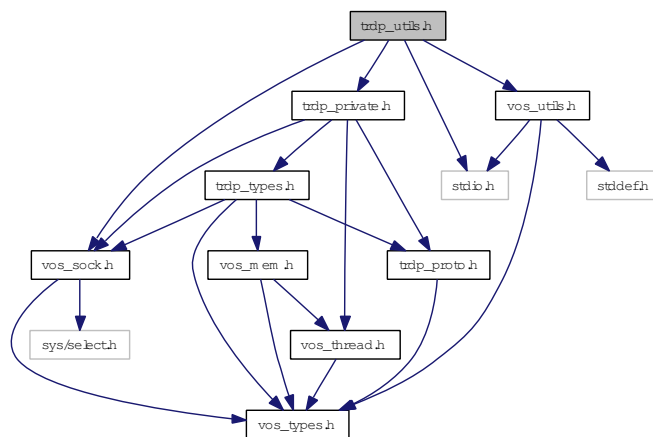
1 if found 0 if not found

5.21 trdp_utils.h File Reference

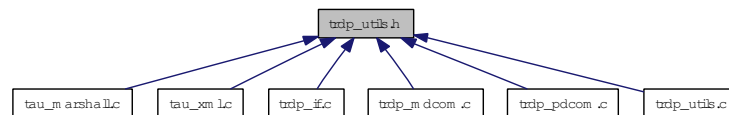
Common utilities for TRDP communication.

```
#include <stdio.h>
#include "trdp_private.h"
#include "vos_utils.h"
#include "vos_sock.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_queueFindComId (PD_ELE_T *pHead, UINT32 comId)`
Return the element with same comId.
- `PD_ELE_T * trdp_queueFindSubAddr (PD_ELE_T *pHead, TRDP_ADDRESSES_T *pAddr)`
Return the element with same comId and IP addresses.
- `MD_ELE_T * trdp_MDqueueFindAddr (MD_ELE_T *pHead, TRDP_ADDRESSES_T *addr)`
Return the element with same comId from MD queue.
- `PD_ELE_T * trdp_queueFindPubAddr (PD_ELE_T *pHead, TRDP_ADDRESSES_T *addr)`
Return the element with same comId and IP addresses.

- void `trdp_queueDelElement` (`PD_ELE_T **pHead`, `PD_ELE_T *pDelete`)
Delete an element.
- void `trdp_MDqueueDelElement` (`MD_ELE_T **ppHead`, `MD_ELE_T *pDelete`)
Delete an element from MD queue.
- void `trdp_MDqueueAppLast` (`MD_ELE_T **pHead`, `MD_ELE_T *pNew`)
Append an element at end of queue.
- void `trdp_MDqueueInsFirst` (`MD_ELE_T **ppHead`, `MD_ELE_T *pNew`)
Insert an element at front of MD queue.
- void `trdp_queueAppLast` (`PD_ELE_T **pHead`, `PD_ELE_T *pNew`)
Append an element at end of queue.
- void `trdp_queueInsFirst` (`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.
- void `trdp_initUncompletedTCP` (`TRDP_APP_SESSION_T appHandle`)
???
- `TRDP_ERR_T trdp_requestSocket` (`TRDP_SOCKETS_T iface[]`, `UINT32 port`, `const TRDP_SEND_PARAM_T *params`, `TRDP_IP_ADDR_T srcIP`, `TRDP_IP_ADDR_T mcGroup`, `TRDP_SOCK_TYPE_T usage`, `TRDP_OPTION_T options`, `BOOL rcvMostly`, `INT32 useSocket`, `INT32 *pIndex`, `TRDP_IP_ADDR_T cornerIp`)
Handle the socket pool: Request a socket from our socket pool.
- void `trdp_releaseSocket` (`TRDP_SOCKETS_T iface[]`, `INT32 lIndex`, `UINT32 connectTimeout`, `BOOL checkAll`)
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.
- `UINT32 trdp_packetSizeMD` (`UINT32 dataSize`)
Get the packet size from the raw data size.
- `UINT32 trdp_getSeqCnt` (`UINT32 comID`, `TRDP_MSG_T msgType`, `TRDP_IP_ADDR_T srcIP`)
Get the initial sequence counter for the comID/message type and subnet (source IP).
- `BOOL trdp_isRcvSeqCnt` (`UINT32 seqCnt`, `UINT32 comId`, `TRDP_MSG_T msgType`, `TRDP_IP_ADDR_T srcIP`)
Check if the sequence counter for the comID/message type and subnet (source IP) has already been received.
- `BOOL trdp_isAddressed` (`const TRDP_URI_USER_T listUri`, `const TRDP_URI_USER_T destUri`)
Check if listener URI is in addressing range of destination URI.

5.21.1 Detailed Description

Common utilities for TRDP communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[trdp_utils.h](#) 962 2013-06-14 08:06:30Z bloehr

5.21.2 Function Documentation

5.21.2.1 int am_big_endian ()

Determine if we are Big or Little endian.

Return values:

!= 0 we are big endian

0 we are little endian

5.21.2.2 UINT32 trdp_getSeqCnt (UINT32 *comId*, TRDP_MSG_T *msgType*, TRDP_IP_ADDR_T *srcIpAddr*)

Get the initial sequence counter for the comID/message type and subnet (source IP).

If the comID/srcIP is not found elsewhere, return 0 - else return its current sequence number (the redundant packet needs the same seqNo)

Note: The standard demands that sequenceCounter is managed per comID/msgType at each publisher, but shall be the same for redundant telegrams (subnet/srcIP).

Parameters:

← ***comId*** comID to look for

← ***msgType*** PD/MD type

← ***srcIpAddr*** Source IP address

Return values:

return the sequence number

Here is the call graph for this function:



5.21.2.3 void trdp_initSockets (TRDP_SOCKETS_T *iface*[])

Handle the socket pool: Initialize it.

Parameters:

← *iface* pointer to the socket pool

5.21.2.4 void trdp_initUncompletedTCP (TRDP_APP_SESSION_T *appHandle*)

???

Parameters:

← *appHandle* session handle

5.21.2.5 BOOL trdp_isAddressed (const TRDP_URI_USER_T *listUri*, const TRDP_URI_USER_T *destUri*)

Check if listener URI is in addressing range of destination URI.

Parameters:

← *listUri* Null terminated listener URI string to compare

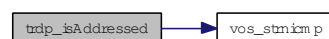
← *destUri* Null terminated destination URI string to compare

Return values:

FALSE - not in addressing range

TRUE - listener URI is in addressing range of destination URI

Here is the call graph for this function:

**5.21.2.6 BOOL trdp_isRcvSeqCnt (UINT32 *seqCnt*, UINT32 *comId*, TRDP_MSG_T *msgType*, TRDP_IP_ADDR_T *srcIP*)**

Check if the sequence counter for the comID/message type and subnet (source IP) has already been received.

Note: The standard demands that sequenceCounter is managed per comID/msgType at each publisher, but shall be the same for redundant telegrams (subnet/srcIP).

Parameters:

← *seqCnt* sequence counter received

← *comId* comID to look for

← *msgType* PD/MD type

← *srcIP* Source IP address

Return values:

return the sequence number

Here is the call graph for this function:



5.21.2.7 void trdp_MDqueueAppLast (MD_ELE_T ** *ppHead*, MD_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

5.21.2.8 void trdp_MDqueueDelElement (MD_ELE_T ** *ppHead*, MD_ELE_T * *pDelete*)

Delete an element from MD queue.

Parameters:

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

5.21.2.9 MD_ELE_T* trdp_MDqueueFindAddr (MD_ELE_T * *pHead*, TRDP_ADDRESSES_T * *addr*)

Return the element with same comId from MD queue.

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

5.21.2.10 void trdp_MDqueueInsFirst (MD_ELE_T ** *ppHead*, MD_ELE_T * *pNew*)

Insert an element at front of MD queue.

Parameters:

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

5.21.2.11 `UINT32 trdp_packetSizeMD (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

5.21.2.12 `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

5.21.2.13 `void trdp_queueAppLast (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

5.21.2.14 `void trdp_queueDelElement (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

5.21.2.15 `PD_ELE_T* trdp_queueFindComId (PD_ELE_T * pHead, UINT32 comId)`

Return the element with same comId.

Parameters:

← *pHead* pointer to head of queue

← *comId* ComID to search for

Return values:

!= NULL pointer to PD element

NULL No PD element found

5.21.2.16 PD_ELE_T* trdp_queueFindPubAddr (PD_ELE_T * *pHead*, TRDP_ADDRESSES_T * *addr*)

Return the element with same comId and IP addresses.

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

5.21.2.17 PD_ELE_T* trdp_queueFindSubAddr (PD_ELE_T * *pHead*, TRDP_ADDRESSES_T * *addr*)

Return the element with same comId and IP addresses.

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

5.21.2.18 void trdp_queueInsFirst (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

5.21.2.19 void trdp_releaseSocket (TRDP_SOCKETS_T *iface*[], INT32 *lIndex*, UINT32 *connectTimeout*, BOOL *checkAll*)

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

Parameters:

- ↔ *iface* socket pool
- ← *lIndex* index of socket to release
- ← *connectTimeout* timeout value

← *checkAll* release all TCP pending sockets

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

In Udp, Release a socket from our socket pool

Parameters:

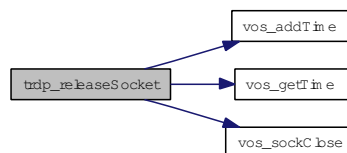
↔ *iface* socket pool

← *Index* index of socket to release

← *connectTimeout* time out

← *checkAll* release all TCP pending sockets

Here is the call graph for this function:



5.21.2.20 TRDP_ERR_T trdp_requestSocket (TRDP_SOCKETS_T *iface*[], UINT32 *port*, const TRDP_SEND_PARAM_T * *params*, TRDP_IP_ADDR_T *srcIP*, TRDP_IP_ADDR_T *mcGroup*, TRDP SOCK_TYPE_T *usage*, TRDP_OPTION_T *options*, BOOL *rcvMostly*, INT32 *useSocket*, INT32 * *pIndex*, TRDP_IP_ADDR_T *cornerIp*)

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

Parameters:

↔ *iface* socket pool

← *port* port to use

← *params* parameters to use

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

← *mcGroup* MC group to join (0 = do not join)

← *usage* type and port to bind to

← *options* blocking/nonblocking

← *rcvMostly* only used for receiving

→ *useSocket* socket to use, do not open a new one

→ *pIndex* returned index of socket pool

← *cornerIp* only used for receiving

Return values:

TRDP_NO_ERR

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

If a multicast group should be joined, we do that on an otherwise suitable socket - up to 20 multicast groups can be joined per socket. If a socket for multicast publishing is requested, we also use the source IP to determine the interface for outgoing multicast traffic.

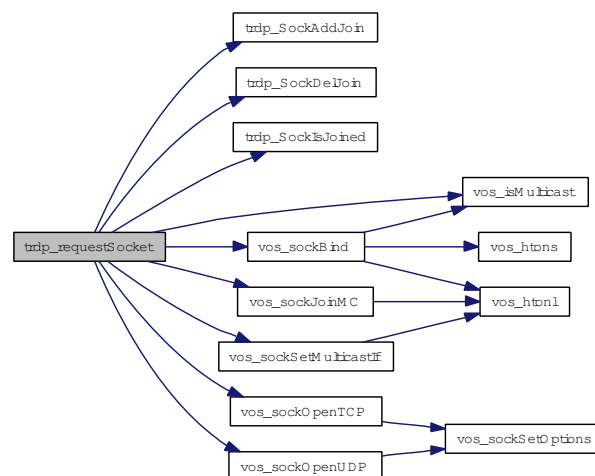
Parameters:

- ↔ **iface** socket pool
- ← **port** port to use
- ← **params** parameters to use
- ← **srcIP** IP to bind to (0 = any address)
- ← **mcGroup** MC group to join (0 = do not join)
- ← **usage** type and port to bind to (PD, MD/UDP, MD/TCP)
- ← **options** blocking/nonblocking
- ← **rcvMostly** primarily used for receiving (tbd: bind on sender, too?)
- **useSocket** socket to use, do not open a new one
- **pIndex** returned index of socket pool
- ← **cornerIp** only used for receiving

Return values:

- TRDP_NO_ERR**
- TRDP_PARAM_ERR**

Here is the call graph for this function:

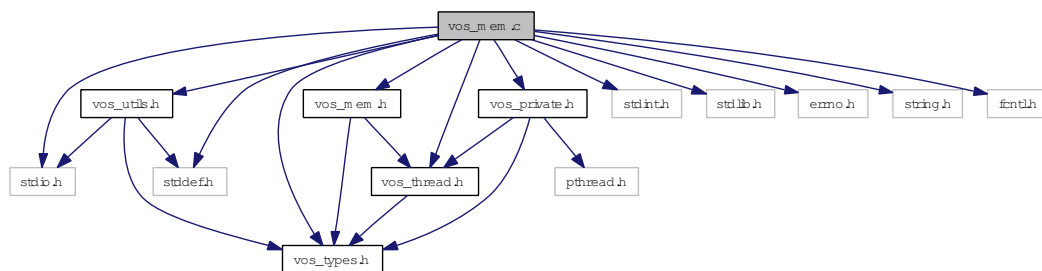


5.22 vos_mem.c File Reference

Memory functions.

```
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.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 "vos_private.h"
```

Include dependency graph for vos_mem.c:



Functions

- [VOS_ERR_T vos_mutexLocalCreate](#) (struct VOS_MUTEX *pMutex)
Create a recursive mutex.
- void [vos_mutexLocalDelete](#) (struct VOS_MUTEX *pMutex)
Delete a mutex.
- EXT_DECL [VOS_ERR_T vos_memInit](#) (UINT8 *pMemoryArea, UINT32 size, const UINT32 fragMem[VOS_MEM_NBLOCKSIZES])
Initialize the memory unit.
- EXT_DECL void [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 void [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 *pMinFree, UINT32 *pNumAllocBlocks, UINT32 *pNumAllocErr, UINT32 *pNumFreeErr, UINT32 blockSize[VOS_MEM_NBLOCKSIZES], UINT32 usedBlockSize[VOS_MEM_NBLOCKSIZES])
Return used and available memory (of memory area above).
- EXT_DECL void [vos_qsort](#) (void *pBuf, UINT32 num, UINT32 size, int(*compare)(const void *, const void *))
Sort an array.
- EXT_DECL void * [vos_bsearch](#) (const void *pKey, const void *pBuf, UINT32 num, UINT32 size, int(*compare)(const void *, const void *))
Binary search in a sorted array.
- EXT_DECL INT32 [vos_strnicmp](#) (const CHAR8 *pStr1, const CHAR8 *pStr2, UINT32 count)
Case insensitive string compare.
- EXT_DECL void [vos_strncpy](#) (CHAR8 *pStrDst, const CHAR8 *pStrSrc, UINT32 count)
String copy with length limitation.
- EXT_DECL [VOS_ERR_T](#) [vos_queueCreate](#) ([VOS_QUEUE_POLICY_T](#) queueType, UINT32 maxNoOfMsg, [VOS_QUEUE_T](#) *pQueueHandle)
Initialize a message queue.
- EXT_DECL [VOS_ERR_T](#) [vos_queueSend](#) ([VOS_QUEUE_T](#) queueHandle, UINT8 *pData, UINT32 size)
Send a message.
- EXT_DECL [VOS_ERR_T](#) [vos_queueReceive](#) ([VOS_QUEUE_T](#) queueHandle, UINT8 **ppData, UINT32 *pSize, UINT32 usTimeout)
Get a message.
- EXT_DECL [VOS_ERR_T](#) [vos_queueDestroy](#) ([VOS_QUEUE_T](#) queueHandle)
Destroy a message queue.

5.22.1 Detailed Description

Memory functions.

OS abstraction of memory access and control

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[vos_mem.c](#) 951 2013-06-13 13:56:42Z 97025

Changes: BL 2012-12-03: ID 1: "using uninitialized PD_ELE_T.pullIpAddress variable" ID 2: "uninitialized PD_ELE_T newPD → pNext in tlp_subscribe()"

5.22.2 Function Documentation**5.22.2.1 EXT_DECL void* vos_bsearch (const void * *pKey*, const void * *pBuf*, UINT32 *num*, UINT32 *size*, int(*)(const void *, const void *) *compare*)**

Binary search in a sorted array.

This is just a wrapper for the standard bsearch function.

Parameters:

- ← *pKey* Key to search for
- ← *pBuf* Pointer to the array to sort
- ← *num* number of elements
- ← *size* size of one element
- ← *compare* Pointer to compare function return -n if arg1 < arg2, return 0 if arg1 == arg2, return +n if arg1 > arg2 where n is an integer != 0

Return values:

Pointer to found element or NULL

5.22.2.2 EXT_DECL UINT8* vos_memAlloc (UINT32 *size*)

Allocate a block of memory (from memory area above).

Parameters:

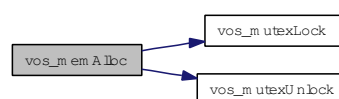
- ← *size* Size of requested block

Return values:

Pointer to memory area

NULL if no memory available

Here is the call graph for this function:



5.22.2.3 EXT_DECL VOS_ERR_T vos_memCount (UINT32 * *pAllocatedMemory*, UINT32 * *pFreeMemory*, UINT32 * *pMinFree*, UINT32 * *pNumAllocBlocks*, UINT32 * *pNumAllocErr*, UINT32 * *pNumFreeErr*, UINT32 *blockSize*[VOS_MEM_NBLOCKSIZES], UINT32 *usedBlockSize*[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
- *pMinFree* Pointer to minimal free memory size in statistics interval
- *pNumAllocBlocks* Pointer to number of allocated memory blocks
- *pNumAllocErr* Pointer to number of allocation errors
- *pNumFreeErr* Pointer to number of free errors
- *blockSize* Pointer to list of memory block sizes
- *usedBlockSize* Pointer to list of used memory blocks

Return values:

- VOS_NO_ERR* no error
- VOS_INIT_ERR* module not initialised

5.22.2.4 EXT_DECL void 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 used

Here is the call graph for this function:



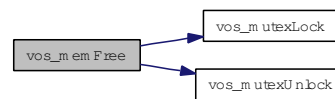
5.22.2.5 EXT_DECL void vos_memFree (void * *pMemBlock*)

Deallocate a block of memory (from memory area above).

Parameters:

- ← *pMemBlock* Pointer to memory block to be freed

Here is the call graph for this function:



5.22.2.6 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_memAlloc and vos_memFree. The used block sizes can be supplied and will be preallocated. If half of the overall size of the requested memory area would be pre-allocated, either by the default pre-allocation table or a provided one, no pre-allocation takes place.

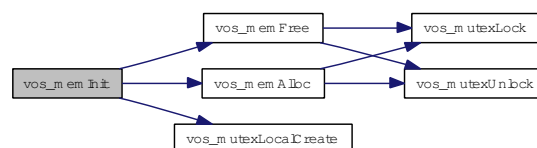
Parameters:

- ← *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
- VOS_MUTEX_ERR** no mutex available

Here is the call graph for this function:



5.22.2.7 VOS_ERR_T vos_mutexLocalCreate (struct VOS_MUTEX * *pMutex*)

Create a recursive mutex.

Fill in a mutex handle. The mutex storage must be already allocated.

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

5.22.2.8 void vos_mutexLocalDelete (struct VOS_MUTEX *pMutex)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← **pMutex** Pointer to mutex struct

5.22.2.9 EXT_DECL void vos_qsort (void *pBuf, UINT32 num, UINT32 size, int(*)(const void *, const void *) compare)

Sort an array.

This is just a wrapper for the standard qsort function.

Parameters:

↔ **pBuf** Pointer to the array to sort

← **num** number of elements

← **size** size of one element

← **compare** Pointer to compare function return -n if arg1 < arg2, return 0 if arg1 == arg2, return +n if arg1 > arg2 where n is an integer != 0

Return values:

none

5.22.2.10 EXT_DECL VOS_ERR_T vos_queueCreate (VOS_QUEUE_POLICY_T queueType, UINT32 maxNoOfMsg, VOS_QUEUE_T *pQueueHandle)

Initialize a message queue.

Returns a handle for further calls

Parameters:

← **queueType** Define queue type (1 = FIFO, 2 = LIFO, 3 = PRIO)

← **maxNoOfMsg** Maximum number of messages

→ **pQueueHandle** Handle of created queue

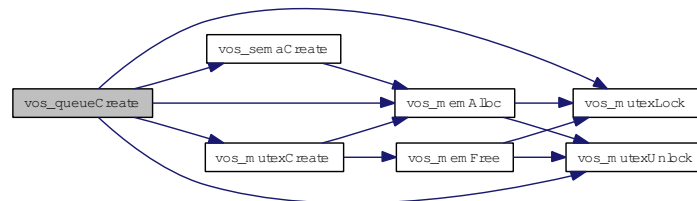
Return values:

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

Here is the call graph for this function:



5.22.2.11 EXT_DECL VOS_ERR_T vos_queueDestroy (VOS_QUEUE_T *queueHandle*)

Destroy a message queue.

Free all resources used by this queue

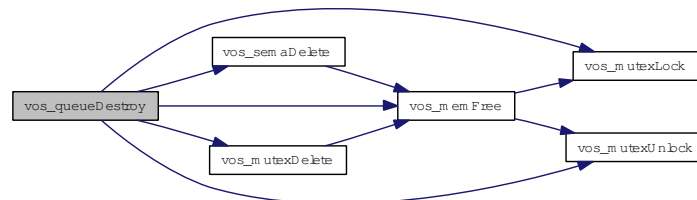
Parameters:

← *queueHandle* 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

Here is the call graph for this function:



5.22.2.12 EXT_DECL VOS_ERR_T vos_queueReceive (VOS_QUEUE_T *queueHandle*, UINT8 ***ppData*, UINT32 **pSize*, UINT32 *usTimeout*)

Get a message.

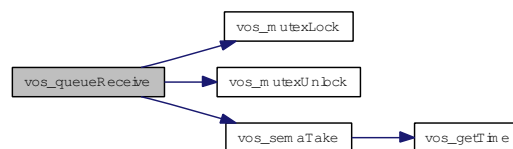
Parameters:

- ← *queueHandle* Queue handle
- *ppData* Pointer to data pointer to be received
- *pSize* Size of receive data
- ← *usTimeout* Maximum time to wait for a message (in usec)

Return values:

- VOSNO_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

Here is the call graph for this function:



5.22.2.13 EXT_DECL VOS_ERR_T vos_queueSend (VOS_QUEUE_T queueHandle, UINT8 * pData, UINT32 size)

Send a message.

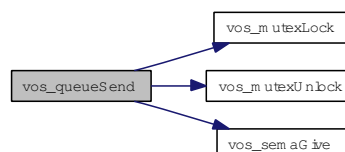
Parameters:

- ← *queueHandle* Queue handle
- ← *pData* Pointer to data to be sent
- ← *size* Size of data to be sent

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

Here is the call graph for this function:



5.22.2.14 EXT_DECL void vos_strncpy (CHAR8 * *pStrDst*, const CHAR8 * *pStrSrc*, UINT32 *count*)

String copy with length limitation.

Parameters:

- ← *pStrDst* Destination string
- ← *pStrSrc* Null terminated string to copy
- ← *count* Maximum number of characters to copy

Return values:

none

5.22.2.15 EXT_DECL INT32 vos_strnicmp (const CHAR8 * *pStr1*, const CHAR8 * *pStr2*, UINT32 *count*)

Case insensitive string compare.

Parameters:

- ← *pStr1* Null terminated string to compare
- ← *pStr2* Null terminated string to compare
- ← *count* Maximum number of characters to compare

Return values:

- 0* - equal
- <0* - string1 less than string 2
- >0* - string 1 greater than string 2

Functions

- EXT_DECL [VOS_ERR_T](#) [vos_memInit](#) (UINT8 *pMemoryArea, UINT32 size, const UINT32 fragMem[VOS_MEM_NBLOCKSIZES])
Initialize the memory unit.
- EXT_DECL void [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 void [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 *pMinFree, UINT32 *pNumAllocBlocks, UINT32 *pNumAllocErr, UINT32 *pNumFreeErr, UINT32 blockSize[VOS_MEM_NBLOCKSIZES], UINT32 usedBlockSize[VOS_MEM_NBLOCKSIZES])
Return used and available memory (of memory area above).
- EXT_DECL void [vos_qsort](#) (void *pBuf, UINT32 num, UINT32 size, int(*compare)(const void *, const void *))
Sort an array.
- EXT_DECL void * [vos_bsearch](#) (const void *pKey, const void *pBuf, UINT32 num, UINT32 size, int(*compare)(const void *, const void *))
Binary search in a sorted array.
- EXT_DECL INT32 [vos_strnicmp](#) (const CHAR8 *pStr1, const CHAR8 *pStr2, UINT32 count)
Case insensitive string compare.
- EXT_DECL void [vos_strncpy](#) (CHAR8 *pStr1, const CHAR8 *pStr2, UINT32 count)
String copy with length limitation.
- EXT_DECL [VOS_ERR_T](#) [vos_queueCreate](#) ([VOS_QUEUE_POLICY_T](#) queueType, UINT32 maxNoOfMsg, [VOS_QUEUE_T](#) *pQueueHandle)
Initialize a message queue.
- EXT_DECL [VOS_ERR_T](#) [vos_queueSend](#) ([VOS_QUEUE_T](#) queueHandle, UINT8 *pData, UINT32 size)
Send a message.
- EXT_DECL [VOS_ERR_T](#) [vos_queueReceive](#) ([VOS_QUEUE_T](#) queueHandle, UINT8 **ppData, UINT32 *pSize, UINT32 usTimeout)
Get a message.
- EXT_DECL [VOS_ERR_T](#) [vos_queueDestroy](#) ([VOS_QUEUE_T](#) queueHandle)
Destroy a message queue.

5.23.1 Detailed Description

Memory and queue functions for OS abstraction.

This module provides memory control supervision

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH Peter Brander (Memory scheme)

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[vos_mem.h](#) 951 2013-06-13 13:56:42Z 97025

5.23.2 Define Documentation

5.23.2.1 #define VOS_MEM_BLOCKSIZEs

Value:

```
{32, 48, 128, 180, 256, 512, 1024, 1480, 2048, \
    4096, 11520, 16384, 32768, 65536, 131072}
```

We internally allocate memory always by these block sizes.

The largest available block is 524288 Bytes, provided the overall size of the used memory allocation area is larger.

5.23.2.2 #define VOS_MEM_PREALLOCATE {0, 0, 0, 0, 0, 0, 8, 0, 0, 1, 0, 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.

5.23.3 Function Documentation

5.23.3.1 EXT_DECL void* vos_bsearch (const void * *pKey*, const void * *pBuf*, UINT32 *num*, UINT32 *size*, int(*)(const void *, const void *) *compare*)

Binary search in a sorted array.

This is just a wrapper for the standard bsearch function.

Parameters:

← *pKey* Key to search for

- ← *pBuf* Pointer to the array to sort
- ← *num* number of elements
- ← *size* size of one element
- ← *compare* Pointer to compare function return -n if arg1 < arg2, return 0 if arg1 == arg2, return +n if arg1 > arg2 where n is an integer != 0

Return values:

Pointer to found element or NULL

5.23.3.2 EXT_DECL UINT8* vos_memAlloc (UINT32 size)

Allocate a block of memory (from memory area above).

Parameters:

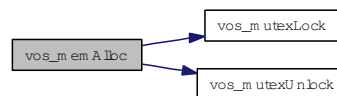
- ← *size* Size of requested block

Return values:

Pointer to memory area

NULL if no memory available

Here is the call graph for this function:


5.23.3.3 EXT_DECL VOS_ERR_T vos_memCount (UINT32 *pAllocatedMemory, UINT32 *pFreeMemory, UINT32 *pMinFree, UINT32 *pNumAllocBlocks, UINT32 *pNumAllocErr, UINT32 *pNumFreeErr, UINT32 blockSize[VOS_MEM_NBLOCKSIZES], UINT32 usedBlockSize[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
- *pMinFree* Pointer to minimal free memory size in statistics interval
- *pNumAllocBlocks* Pointer to number of allocated memory blocks
- *pNumAllocErr* Pointer to number of allocation errors
- *pNumFreeErr* Pointer to number of free errors
- *blockSize* Pointer to list of memory block sizes
- *usedBlockSize* Pointer to list of used memory blocks

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

5.23.3.4 EXT_DECL void 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

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 used

Here is the call graph for this function:



5.23.3.5 EXT_DECL void vos_memFree (void * *pMemBlock*)

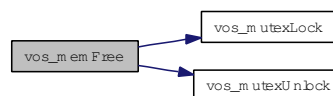
Deallocate a block of memory (from memory area above).

Parameters:

← *pMemBlock* Pointer to memory block to be freed

← *pMemBlock* Pointer to memory block to be freed

Here is the call graph for this function:



5.23.3.6 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_memAlloc and vos_memFree. The used block sizes can be supplied and will be preallocated. If half of the overall size of the requested memory area would be pre-allocated, either by the default pre-allocation table or a provided one, no pre-allocation takes place.

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:

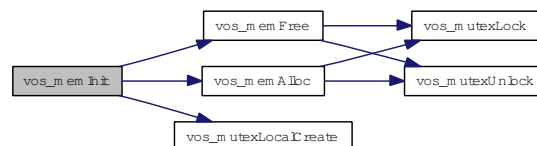
VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

VOS_MEM_ERR no memory available

VOS_MUTEX_ERR no mutex available

Here is the call graph for this function:



5.23.3.7 EXT_DECL void vos_qsort (void *pBuf, UINT32 num, UINT32 size, int(*)(const void *, const void *) compare)

Sort an array.

This is just a wrapper for the standard qsort function.

Parameters:

↔ **pBuf** Pointer to the array to sort

← **num** number of elements

← **size** size of one element

← **compare** Pointer to compare function return -n if arg1 < arg2, return 0 if arg1 == arg2, return +n if arg1 > arg2 where n is an integer != 0

Return values:

none

5.23.3.8 EXT_DECL VOS_ERR_T vos_queueCreate (VOS_QUEUE_POLICY_T *queueType*, UINT32 *maxNoOfMsg*, VOS_QUEUE_T * *pQueueHandle*)

Initialize a message queue.

Returns a handle for further calls

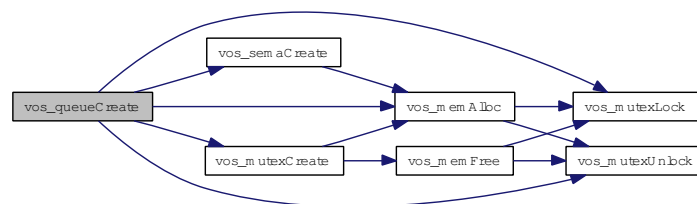
Parameters:

- ← *queueType* Define queue type (1 = FIFO, 2 = LIFO, 3 = PRIO)
- ← *maxNoOfMsg* Maximum number of messages
- *pQueueHandle* Handle of created queue

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

Here is the call graph for this function:



5.23.3.9 EXT_DECL VOS_ERR_T vos_queueDestroy (VOS_QUEUE_T *queueHandle*)

Destroy a message queue.

Free all resources used by this queue

Parameters:

- ← *queueHandle* 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

Free all resources used by this queue

Parameters:

← *queueHandle* Queue handle

Return values:

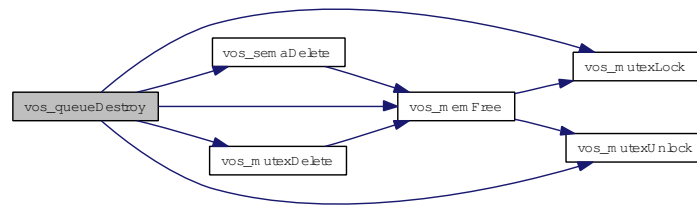
VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

Here is the call graph for this function:



5.23.3.10 EXT_DECL VOS_ERR_T vos_queueReceive (VOS_QUEUE_T queueHandle, UINT8 **ppData, UINT32 *pSize, UINT32 usTimeout)

Get a message.

Parameters:

← *queueHandle* Queue handle

→ *ppData* Pointer to data pointer to be received

→ *pSize* Size of receive data

← *usTimeout* Maximum time to wait for a message (in usec)

Return values:

VOSNO_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

Parameters:

← *queueHandle* Queue handle

→ *ppData* Pointer to data pointer to be received

→ *pSize* Size of receive data

← *usTimeout* Maximum time to wait for a message (in usec)

Return values:

VOSNO_ERR no error

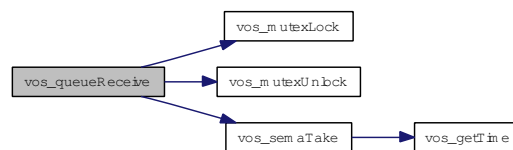
VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_QUEUE_ERR queue is empty

Here is the call graph for this function:



5.23.3.11 EXT_DECL VOS_ERR_T vos_queueSend (VOS_QUEUE_T *queueHandle*, UINT8 * *pData*, UINT32 *size*)

Send a message.

Parameters:

← *queueHandle* Queue handle

← *pData* Pointer to data to be sent

← *size* Size of data to be sent

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

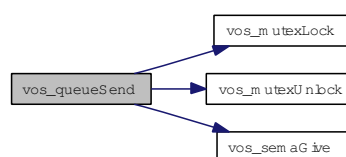
VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_INIT_ERR not supported

VOS_QUEUE_ERR error creating queue

Here is the call graph for this function:



5.23.3.12 EXT_DECL void vos_strncpy (CHAR8 * *pStrDst*, const CHAR8 * *pStrSrc*, UINT32 *count*)

String copy with length limitation.

Parameters:

- ← *pStrDst* Destination string
- ← *pStrSrc* Null terminated string to copy
- ← *count* Maximum number of characters to copy

Return values:

none

5.23.3.13 EXT_DECL INT32 vos_strnicmp (const CHAR8 * *pStr1*, const CHAR8 * *pStr2*, UINT32 *count*)

Case insensitive string compare.

Parameters:

- ← *pStr1* Null terminated string to compare
- ← *pStr2* Null terminated string to compare
- ← *count* Maximum number of characters to compare

Return values:

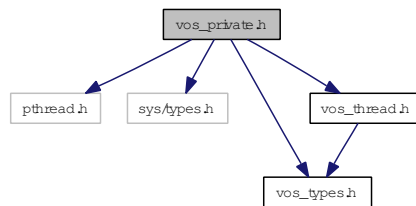
- 0* - equal
- <0* - string1 less than string 2
- >0* - string 1 greater than string 2

5.24 vos_private.h File Reference

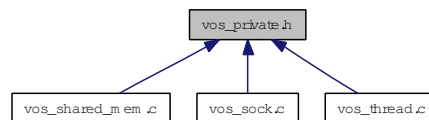
Private definitions for the OS abstraction layer.

```
#include <pthread.h>
#include <sys/types.h>
#include "vos_types.h"
#include "vos_thread.h"
```

Include dependency graph for posix/vos_private.h:



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



Functions

- [VOS_ERR_T vos_mutexLocalCreate](#) (struct VOS_MUTEX *pMutex)
Create a recursive mutex.
- void [vos_mutexLocalDelete](#) (struct VOS_MUTEX *pMutex)
Delete a mutex.

5.24.1 Detailed Description

Private definitions for the OS abstraction layer.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

vos_private.h 951 2013-06-13 13:56:42Z 97025

5.24.2 Function Documentation**5.24.2.1 VOS_ERR_T vos_mutexLocalCreate (struct VOS_MUTEX * *pMutex*)**

Create a recursive mutex.

Fill in a mutex handle. The mutex storage must be already allocated.

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

5.24.2.2 void vos_mutexLocalDelete (struct VOS_MUTEX * *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

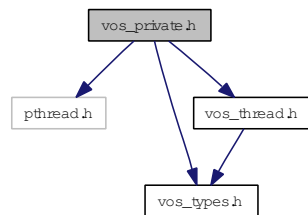
← *pMutex* Pointer to mutex struct

5.25 vos_private.h File Reference

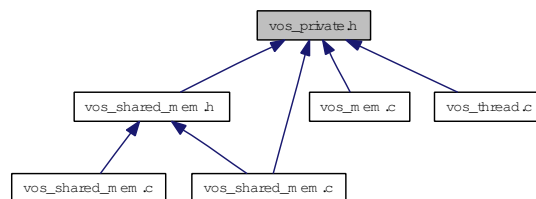
Private definitions for the OS abstraction layer.

```
#include <pthread.h>
#include "vos_types.h"
#include "vos_thread.h"
```

Include dependency graph for windows/vos_private.h:



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



Functions

- [VOS_ERR_T vos_mutexLocalCreate](#) (struct VOS_MUTEX *pMutex)
Create a recursive mutex.
- void [vos_mutexLocalDelete](#) (struct VOS_MUTEX *pMutex)
Delete a mutex.

5.25.1 Detailed Description

Private definitions for the OS abstraction layer.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

vos_private.h 951 2013-06-13 13:56:42Z 97025

5.25.2 Function Documentation**5.25.2.1 VOS_ERR_T vos_mutexLocalCreate (struct VOS_MUTEX * *pMutex*)**

Create a recursive mutex.

Fill in a mutex handle. The mutex storage must be already allocated.

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

5.25.2.2 void vos_mutexLocalDelete (struct VOS_MUTEX * *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

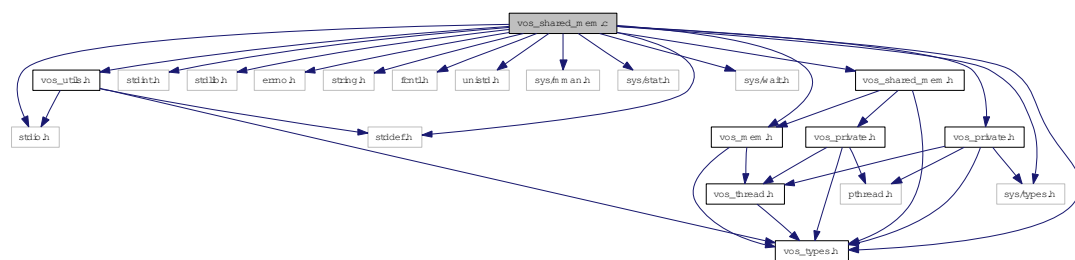
← *pMutex* Pointer to mutex struct

5.26 vos_shared_mem.c File Reference

Shared Memory functions.

```
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "vos_types.h"
#include "vos_mem.h"
#include "vos_utils.h"
#include "vos_private.h"
#include "vos_shared_mem.h"
```

Include dependency graph for posix/vos_shared_mem.c:



Functions

- 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.

5.26.1 Detailed Description

Shared Memory functions.

OS abstraction of Shared memory access and control

Note:

Project: TCNOpen TRDP prototype stack

Author:

Kazumasa Aiba, TOSHIBA

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright TOSHIBA, Japan, 2013.

Id

[vos_mem.h](#) 282 2013-01-11 07:08:44Z 97029

5.26.2 Function Documentation

5.26.2.1 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_MEM_ERR no memory available

5.26.2.2 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)

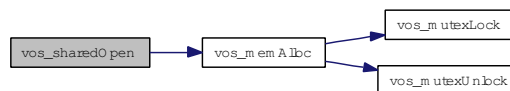
- *pHandle* Pointer to returned handle
- *ppMemoryArea* Pointer to pointer to memory area
- ↔ *pSize* Pointer to size of area to allocate, on return actual size after attach

Return values:

VOS_NO_ERR no error

VOS_MEM_ERR no memory available

Here is the call graph for this function:

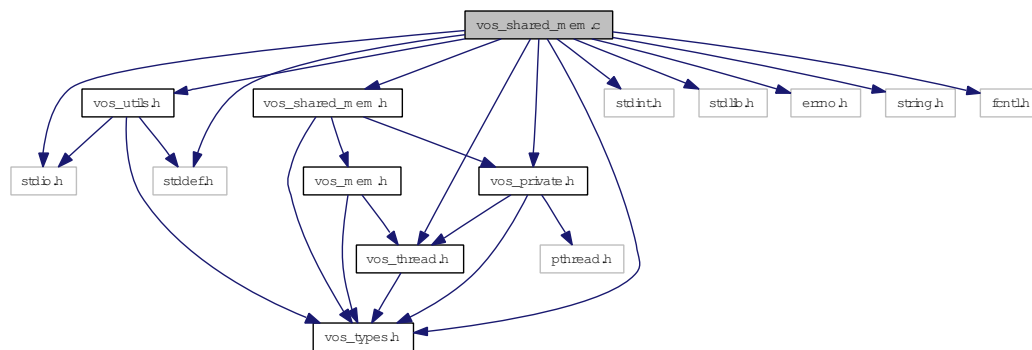


5.27 vos_shared_mem.c File Reference

Shared Memory functions.

```
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include "vos_types.h"
#include "vos_utils.h"
#include "vos_shared_mem.h"
#include "vos_thread.h"
#include "vos_private.h"
```

Include dependency graph for windows/vos_shared_mem.c:



Functions

- 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.

5.27.1 Detailed Description

Shared Memory functions.

OS abstraction of Shared memory access and control

Note:

Project: TCNOpen TRDP prototype stack

Author:

Kazumasa Aiba, TOSHIBA

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright TOSHIBA, Japan, 2013.

Id

[vos_mem.h](#) 282 2013-01-11 07:08:44Z 97029

5.27.2 Function Documentation

5.27.2.1 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_MEM_ERR* no memory available

5.27.2.2 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:

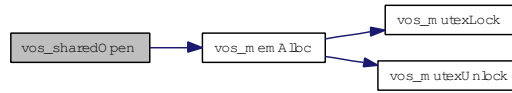
- ← *pKey* Unique identifier (file name)
- *pHandle* Pointer to returned handle
- *ppMemoryArea* Pointer to pointer to memory area
- ↔ *pSize* Pointer to size of area to allocate, on return actual size after attach

Return values:

- VOS_NO_ERR* no error

VOS_MEM_ERR no memory available

Here is the call graph for this function:

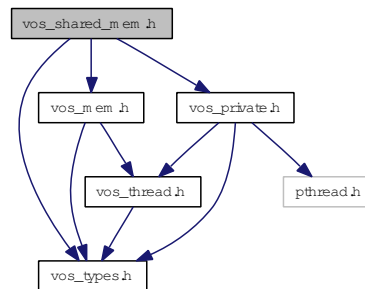


5.28 vos_shared_mem.h File Reference

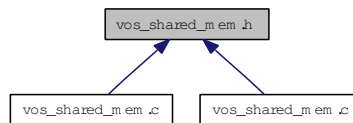
Shared Memory functions for OS abstraction.

```
#include "vos_types.h"
#include "vos_mem.h"
#include "vos_private.h"
```

Include dependency graph for vos_shared_mem.h:



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



Functions

- 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.

5.28.1 Detailed Description

Shared Memory functions for OS abstraction.

This module provides shared memory control supervision

Note:

Project: TCNOpen TRDP prototype stack

Author:

Kazumasa Aiba, TOSHIBA

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright TOSHIBA, Japan, 2013.

Id

[vos_mem.h](#) 282 2013-01-11 07:08:44Z 97029

5.28.2 Function Documentation**5.28.2.1 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_MEM_ERR* no memory available

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_MEM_ERR* no memory available

5.28.2.2 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_MEM_ERR* no memory available

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.

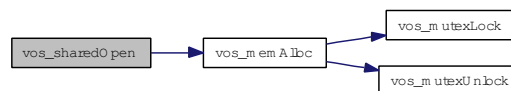
Parameters:

- ← *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_MEM_ERR* no memory available

Here is the call graph for this function:

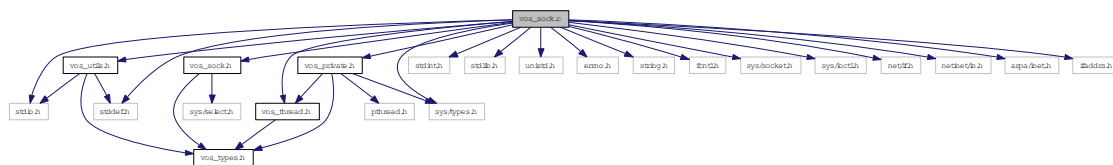


5.29 vos_sock.c File Reference

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 <net/if.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <ifaddrs.h>
#include "vos_utils.h"
#include "vos_sock.h"
#include "vos_thread.h"
#include "vos_private.h"
```

Include dependency graph for posix/vos_sock.c:



Functions

- **BOOL** [vos_getMacAddress](#) (UINT8 *pMacAddr, const char *pIfName)
Get the MAC address for a named interface.
- **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 UINT32 [vos_dottedIP](#) (const CHAR8 *pDottedIP)
Convert IP address from dotted dec.
- EXT_DECL const CHAR8 * [vos_ipDotted](#) (UINT32 ipAddress)
Convert IP address to dotted dec.
- EXT_DECL BOOL [vos_isMulticast](#) (UINT32 ipAddress)
Check if the supplied address is a multicast group address.
- EXT_DECL INT32 [vos_select](#) (INT32 highDesc, VOS_FDS_T *pReadableFD, VOS_FDS_T *pWriteableFD, VOS_FDS_T *pErrorFD, [VOS_TIME_T](#) *pTimeOut)
select function.
- EXT_DECL [VOS_ERR_T](#) [vos_getInterfaces](#) (UINT32 *pAddrCnt, VOS_IF_REC_T ifAddrs[])
Get a list of interface addresses The caller has to provide an array of interface records to be filled.
- EXT_DECL [VOS_ERR_T](#) [vos_sockInit](#) (void)
Initialize the socket library.
- EXT_DECL [VOS_ERR_T](#) [vos_sockGetMAC](#) (UINT8 pMAC[VOS_MAC_SIZE])
Return the MAC address of the default adapter.
- EXT_DECL [VOS_ERR_T](#) [vos_sockOpenUDP](#) (INT32 *pSock, const [VOS_SOCKET_OPT_T](#) *pOptions)
Create an UDP socket.
- EXT_DECL [VOS_ERR_T](#) [vos_sockOpenTCP](#) (INT32 *pSock, const [VOS_SOCKET_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_SOCKET_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 *pSize, UINT32 ipAddress, UINT16 port)
Send UDP data.
- EXT_DECL [VOS_ERR_T vos_sockReceiveUDP](#) (INT32 sock, UINT8 *pBuffer, UINT32 *pSize, UINT32 *pSrcIPAddr, UINT16 *pSrcIPPort, UINT32 *pDstIPAddr, BOOL peek)
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 *pIPAddr, 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 *pSize)
Send TCP data.
- EXT_DECL [VOS_ERR_T vos_sockReceiveTCP](#) (INT32 sock, UINT8 *pBuffer, UINT32 *pSize)
Receive TCP data.
- EXT_DECL [VOS_ERR_T vos_sockSetMulticastIf](#) (INT32 sock, UINT32 mcIfAddress)
Set Using Multicast I/F.

5.29.1 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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012-2013.

Id

vos_sock.c 955 2013-06-13 15:29:12Z bloehr

5.29.2 Function Documentation

5.29.2.1 EXT_DECL UINT32 vos_dottedIP (const CHAR8 * *pDottedIP*)

Convert IP address from dotted dec.

to !host! endianness

Parameters:

← *pDottedIP* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

Here is the call graph for this function:



5.29.2.2 EXT_DECL VOS_ERR_T vos_getInterfaces (UINT32 * *pAddrCnt*, VOS_IF_REC_T *ifAddrs*[])

Get a list of interface addresses The caller has to provide an array of interface records to be filled.

Parameters:

↔ *pAddrCnt* in: pointer to array size of interface record out: pointer to number of interface records read

↔ *ifAddrs* array of interface records

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMAC == NULL

Here is the call graph for this function:



5.29.2.3 BOOL vos_getMacAddress (UINT8 * *pMacAddr*, const char * *pIfName*)

Get the MAC address for a named interface.

Parameters:

→ *pMacAddr* pointer to array of MAC address to return

← *pIfName* pointer to interface name

Return values:

TRUE if successfull

5.29.2.4 EXT_DECL UINT32 vos_htonl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.29.2.5 EXT_DECL UINT16 vos_htons (UINT16 *val*)

Byte swapping.

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.29.2.6 EXT_DECL const CHAR8* vos_ipDotted (UINT32 *ipAddress*)

Convert IP address to dotted dec.

from !host! endianness.

Parameters:

← *ipAddress* address in UINT32 in host endianness

Return values:

IP address as dotted decimal.

5.29.2.7 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

5.29.2.8 EXT_DECL UINT32 vos_ntohl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.29.2.9 EXT_DECL UINT16 vos_ntohs (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.29.2.10 EXT_DECL INT32 vos_select (INT32 *highDesc*, VOS_FDS_T * *pReadableFD*, VOS_FDS_T * *pWriteableFD*, VOS_FDS_T * *pErrorFD*, VOS_TIME_T * *pTimeOut*)

select function.

Set the ready sockets in the supplied sets. Note: Some target systems might define this function as NOP.

Parameters:

- ← *highDesc* max. socket descriptor + 1
- ↔ *pReadableFD* pointer to readable socket set
- ↔ *pWriteableFD* pointer to writeable socket set
- ↔ *pErrorFD* pointer to error socket set
- ← *pTimeOut* pointer to time out value

Return values:

number of ready file descriptors

5.29.2.11 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.

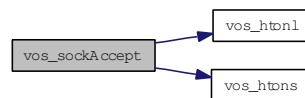
Parameters:

- ← *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:



5.29.2.12 EXT_DECL VOS_ERR_T vos_sockBind (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Bind a socket to an address and port.

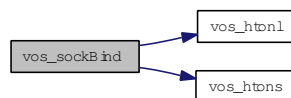
Parameters:

- ← *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:



5.29.2.13 EXT_DECL VOS_ERR_T vos_sockClose (INT32 *sock*)

Close a socket.

Release any resources acquired by this socket

Parameters:

← *sock* socket descriptor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown

5.29.2.14 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

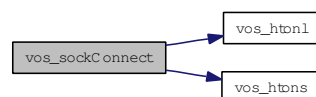
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR Input/Output error

Here is the call graph for this function:



5.29.2.15 EXT_DECL VOS_ERR_T vos_sockGetMAC (UINT8 *pMAC*[VOS_MAC_SIZE])

Return the MAC address of the default adapter.

Parameters:

→ *pMAC* return MAC address.

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMAC == NULL

VOS SOCK_ERR socket not available or option not supported

Here is the call graph for this function:



5.29.2.16 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

5.29.2.17 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

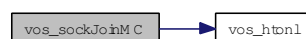
Return values:

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:



5.29.2.18 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.

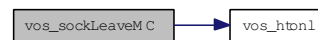
Parameters:

- ← *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:



5.29.2.19 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

5.29.2.20 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:



5.29.2.21 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:



5.29.2.22 EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 sock, UINT8 * pBuffer, UINT32 * 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 occurred. 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

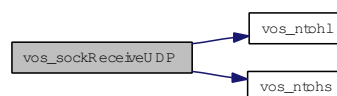
Return values:**VOS_NO_ERR** no error**VOS_PARAM_ERR** sock descriptor unknown, parameter error**VOS_IO_ERR** data could not be read**VOS_NODATA_ERR** no data**VOS_BLOCK_ERR** Call would have blocked in blocking mode**5.29.2.23 EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 *sock*, UINT8 * *pBuffer*,
UINT32 * *pSize*, UINT32 * *pSrcIPAddr*, UINT16 * *pSrcIPPort*, UINT32 * *pDstIPAddr*,
BOOL *peek*)**

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 occurred. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned. If pointers are provided, source IP, source port and destination IP will be reported on return.

Parameters:← *sock* socket descriptor→ *pBuffer* pointer to applications data buffer↔ *pSize* pointer to the received data size→ *pSrcIPAddr* pointer to source IP→ *pSrcIPPort* pointer to source port→ *pDstIPAddr* pointer to dest IP← *peek* if true, leave data in queue**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**VOS_BLOCK_ERR** Call would have blocked in blocking mode

Here is the call graph for this function:



5.29.2.24 EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 * *pSize*)

Send TCP data.

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ↔ *pSize* In: size of the data to send, Out: no of bytes sent

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be sent
- VOS_NOCONN_ERR** no TCP connection
- VOS_BLOCK_ERR** Call would have blocked in blocking mode

5.29.2.25 EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 * *pSize*, UINT32 *ipAddress*, UINT16 *port*)

Send UDP data.

Send data to the supplied address and port.

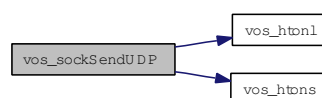
Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ↔ *pSize* In: size of the data to send, Out: no of bytes sent
- ← *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_BLOCK_ERR** Call would have blocked in blocking mode

Here is the call graph for this function:



5.29.2.26 EXT_DECL VOS_ERR_T vos_sockSetMulticastIf (INT32 *sock*, UINT32 *mcIfAddress*)

Set Using Multicast I/F.

Parameters:

- ← *sock* socket descriptor
- ← *mcIfAddress* using Multicast I/F Address

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:



5.29.2.27 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:

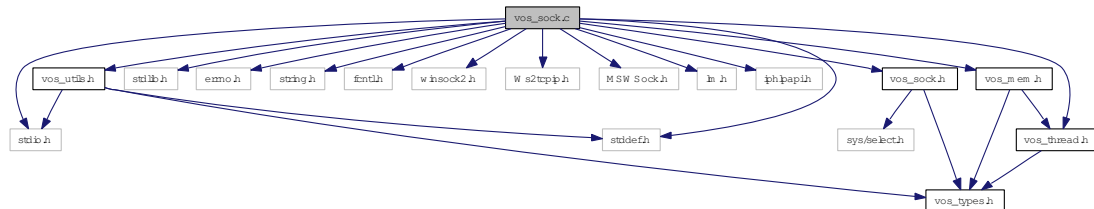
- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown

5.30 vos_sock.c File Reference

Socket functions.

```
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <winsock2.h>
#include <Ws2tcpip.h>
#include <MSWSock.h>
#include <lm.h>
#include <iphlpapi.h>
#include "vos_utils.h"
#include "vos_sock.h"
#include "vos_thread.h"
#include "vos_mem.h"
```

Include dependency graph for windows/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 UINT32 [vos_dottedIP](#) (const CHAR8 *pDottedIP)

Convert IP address from dotted dec.

- EXT_DECL const CHAR8 * [vos_ipDotted](#) (UINT32 ipAddress)
Convert IP address to dotted dec.
- EXT_DECL BOOL [vos_isMulticast](#) (UINT32 ipAddress)
Check if the supplied address is a multicast group address.
- EXT_DECL [VOS_ERR_T](#) [vos_getInterfaces](#) (UINT32 *pAddrCnt, [VOS_IF_REC_T](#) ifAddrs[])
 - Get a list of interface addresses The caller has to provide an array of interface records to be filled.*
- EXT_DECL INT32 [vos_select](#) (INT32 highDesc, [VOS_FDS_T](#) *pReadableFD, [VOS_FDS_T](#) *pWriteableFD, [VOS_FDS_T](#) *pErrorFD, [VOS_TIME_T](#) *pTimeout)
 - select function.*
- EXT_DECL [VOS_ERR_T](#) [vos_sockInit](#) (void)
 - Initialize the socket library.*
- EXT_DECL [VOS_ERR_T](#) [vos_sockGetMAC](#) (UINT8 pMAC[VOS_MAC_SIZE])
 - Return the MAC address of the default adapter.*
- EXT_DECL [VOS_ERR_T](#) [vos_sockOpenUDP](#) (INT32 *pSock, const [VOS_SOCKET_OPT_T](#) *pOptions)
 - Create an UDP socket.*
- EXT_DECL [VOS_ERR_T](#) [vos_sockOpenTCP](#) (INT32 *pSock, const [VOS_SOCKET_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_SOCKET_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 *pSize, UINT32 ipAddress, UINT16 port)
 - Send UDP data.*
- EXT_DECL [VOS_ERR_T](#) [vos_sockReceiveUDP](#) (INT32 sock, UINT8 *pBuffer, UINT32 *pSize, UINT32 *pSrcIPAddr, UINT16 *pSrcIPPort, UINT32 *pDstIPAddr, BOOL peek)

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 *pSize)

Send TCP data.

- EXT_DECL [VOS_ERR_T vos_sockReceiveTCP](#) (INT32 sock, UINT8 *pBuffer, UINT32 *pSize)

Receive TCP data.

- EXT_DECL [VOS_ERR_T vos_sockSetMulticastIf](#) (INT32 sock, UINT32 mcIfAddress)

Set Using Multicast I/F.

5.30.1 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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

vos_sock.c 951 2013-06-13 13:56:42Z 97025

5.30.2 Function Documentation

5.30.2.1 EXT_DECL UINT32 vos_dottedIP (const CHAR8 * *pDottedIP*)

Convert IP address from dotted dec.

to !host! endianness

Parameters:

← *pDottedIP* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

Here is the call graph for this function:



5.30.2.2 EXT_DECL VOS_ERR_T vos_getInterfaces (UINT32 * *pAddrCnt*, VOS_IF_REC_T *ifAddrs*[]))

Get a list of interface addresses The caller has to provide an array of interface records to be filled.

Parameters:

↔ *pAddrCnt* in: pointer to array size of interface record out: pointer to number of interface records read

↔ *ifAddrs* array of interface records

Return values:

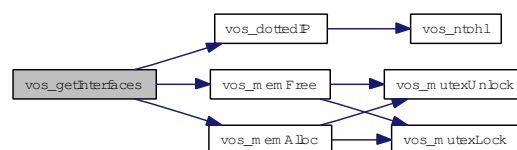
VOS_NO_ERR no error

VOS_PARAM_ERR *pAddrCnt* and/or *ifAddrs* == NULL

VOS_MEM_ERR memory allocation error

VOS SOCK_ERR GetAdaptersInfo() error

Here is the call graph for this function:



5.30.2.3 EXT_DECL UINT32 vos_htonl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.30.2.4 EXT_DECL UINT16 vos_htons (UINT16 *val*)

Byte swapping.

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.30.2.5 EXT_DECL const CHAR8* vos_ipDotted (UINT32 *ipAddress*)

Convert IP address to dotted dec.

from !host! endianness.

Parameters:

← *ipAddress* address in UINT32 in host endianness

Return values:

IP address as dotted decimal.

5.30.2.6 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

5.30.2.7 EXT_DECL UINT32 vos_ntohl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.30.2.8 EXT_DECL UINT16 vos_ntohs (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.30.2.9 EXT_DECL INT32 vos_select (INT32 *highDesc*, VOS_FDS_T * *pReadableFD*, VOS_FDS_T * *pWriteableFD*, VOS_FDS_T * *pErrorFD*, VOS_TIME_T * *pTimeOut*)

select function.

Set the ready sockets in the supplied sets. Note: Some target systems might define this function as NOP.

Parameters:

- ← *highDesc* max. socket descriptor + 1
- ↔ *pReadableFD* pointer to readable socket set
- ↔ *pWriteableFD* pointer to writeable socket set
- ↔ *pErrorFD* pointer to error socket set
- ← *pTimeOut* pointer to time out value

Return values:

number of ready file descriptors

5.30.2.10 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.

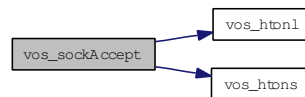
Parameters:

- ← *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:



5.30.2.11 EXT_DECL VOS_ERR_T vos_sockBind (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Bind a socket to an address and port.

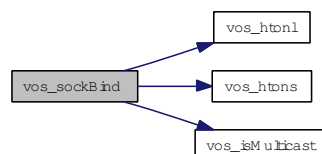
Parameters:

- ← *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:



5.30.2.12 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

5.30.2.13 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:

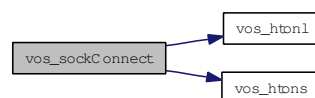
VOS_NO_ERR no error

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:



5.30.2.14 EXT_DECL VOS_ERR_T vos_sockGetMAC (UINT8 *pMAC*[VOS_MAC_SIZE])

Return the MAC address of the default adapter.

Parameters:

→ *pMAC* return MAC address.

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMAC == NULL

VOS_SOCK_ERR socket not available or option not supported

5.30.2.15 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

5.30.2.16 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:

**5.30.2.17 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

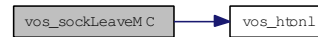
Return values:

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:



5.30.2.18 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

5.30.2.19 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:



5.30.2.20 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:



5.30.2.21 EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 *sock*, UINT8 * *pBuffer*, UINT32 * *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 occurred. 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
- VOS_BLOCK_ERR** call would have blocked in blocking mode

5.30.2.22 EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 *sock*, UINT8 * *pBuffer*, UINT32 * *pSize*, UINT32 * *pSrcIPAddr*, UINT16 * *pSrcIPPort*, UINT32 * *pDstIPAddr*, BOOL *peek*)

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 occurred. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned. If pointers are provided, source IP, source port and destination IP will be reported on return.

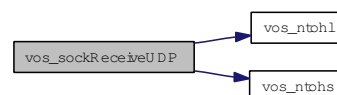
Parameters:

- ← *sock* socket descriptor
- *pBuffer* pointer to applications data buffer
- ↔ *pSize* pointer to the received data size
- *pSrcIPAddr* pointer to source IP
- *pSrcIPPort* pointer to source port
- *pDstIPAddr* pointer to dest IP
- ← *peek* if true, leave data in queue

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
- VOS_BLOCK_ERR** Call would have blocked in blocking mode

Here is the call graph for this function:



5.30.2.23 EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 * *pSize*)

Send TCP data.

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ↔ *pSize* IN: bytes to send, OUT: bytes sent

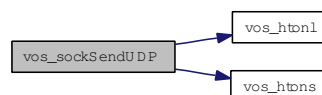
Return values:*VOS_NO_ERR* no error*VOS_PARAM_ERR* sock descriptor unknown, parameter error*VOS_IO_ERR* data could not be sent*VOS_NOCONN_ERR* no TCP connection*VOS_BLOCK_ERR* Call would have blocked in blocking mode**5.30.2.24 EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 *sock*, const UINT8 * *pBuffer*,
UINT32 * *pSize*, 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↔ *pSize* IN: bytes to send, OUT: bytes sent← *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_BLOCK_ERR* Call would have blocked in blocking mode

Here is the call graph for this function:

**5.30.2.25 EXT_DECL VOS_ERR_T vos_sockSetMulticastIf (INT32 *sock*, UINT32 *mcIfAddress*)**

Set Using Multicast I/F.

Parameters:← *sock* socket descriptor← *mcIfAddress* using Multicast I/F Address**Return values:***VOS_NO_ERR* no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

Here is the call graph for this function:



5.30.2.26 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

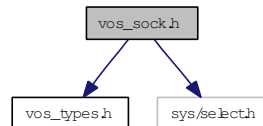
5.31 vos_sock.h File Reference

Typedefs for OS abstraction.

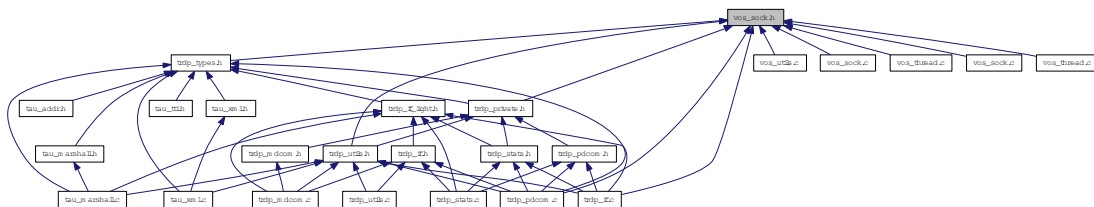
```
#include "vos_types.h"
```

```
#include <sys/select.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 4

The maximum number of sockets influences memory usage; for small systems we should define a smaller set.

- #define VOS_MAX_MULTICAST_CNT 5

The maximum number of multicast groups one socket can join.

- #define VOS_TTL_MULTICAST 64

The maximum number of hops a multicast packet can take.

- #define VOS_MAX_IF_NAME_SIZE 16

The maximum size for the interface name.

- #define VOS_MAX_NUM_IF 4

The maximum number of IP interface adapters that can be handled by VOS.

- #define VOS_MAX_NUM_UNICAST 10

The maximum number of unicast addresses that can be handled by VOS.

- #define **VOS_MAC_SIZE** 6
The MAC size supported by VOS.
- #define **VOS_INVALID_SOCKET** -1
Invalid socket number.

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 UINT32 **vos_dottedIP** (const CHAR8 *pDottedIP)
Convert IP address from dotted dec.
- EXT_DECL const CHAR8 * **vos_ipDotted** (UINT32 ipAddress)
Convert IP address to dotted dec.
- EXT_DECL BOOL **vos_isMulticast** (UINT32 ipAddress)
Check if the supplied address is a multicast group address.
- EXT_DECL **VOS_ERR_T** **vos_getInterfaces** (UINT32 *pAddrCnt, VOS_IF_REC_T ifAddrs[])
 - Get a list of interface addresses The caller has to provide an array of interface records to be filled.*
- EXT_DECL INT32 **vos_select** (INT32 highDesc, VOS_FDS_T *pReadableFD, VOS_FDS_T *pWriteableFD, VOS_FDS_T *pErrorFD, **VOS_TIME_T** *pTimeOut)
 - select function.*
- EXT_DECL **VOS_ERR_T** **vos_sockInit** (void)
 - Initialize the socket library.*
- EXT_DECL **VOS_ERR_T** **vos_sockGetMAC** (UINT8 pMAC[VOS_MAC_SIZE])
 - Return the MAC address of the default adapter.*
- 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_SOCKET_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_SOCKET_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 *pSize, UINT32 ipAddress, UINT16 port)
Send UDP data.
- EXT_DECL [VOS_ERR_T vos_sockReceiveUDP](#) (INT32 sock, UINT8 *pBuffer, UINT32 *pSize, UINT32 *pSrcIPAddr, UINT16 *pSrcIPPort, UINT32 *pDstIPAddr, BOOL peek)
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 *pIPAddr, 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 *pSize)
Send TCP data.
- EXT_DECL [VOS_ERR_T vos_sockReceiveTCP](#) (INT32 sock, UINT8 *pBuffer, UINT32 *pSize)
Receive TCP data.
- EXT_DECL [VOS_ERR_T vos_sockSetMulticastIf](#) (INT32 sock, UINT32 mcIfAddress)
Set Using Multicast I/F.

5.31.1 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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[vos_sock.h](#) 951 2013-06-13 13:56:42Z 97025

5.31.2 Define Documentation

5.31.2.1 #define VOS_MAX_SOCKET_CNT 4

The maximum number of sockets influences memory usage; for small systems we should define a smaller set.

The maximum number of concurrent usable sockets per application session

5.31.3 Function Documentation

5.31.3.1 EXT_DECL UINT32 vos_dottedIP (const CHAR8 * *pDottedIP*)

Convert IP address from dotted dec.

to !host! endianness

Parameters:

← *pDottedIP* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

Here is the call graph for this function:



5.31.3.2 EXT_DECL VOS_ERR_T vos_getInterfaces (UINT32 * *pAddrCnt*, VOS_IF_REC_T *ifAddrs*[])

Get a list of interface addresses The caller has to provide an array of interface records to be filled.

Parameters:

- ↔ *pAddrCnt* in: pointer to array size of interface record out: pointer to number of interface records read
- ↔ *ifAddrs* array of interface records

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** *pAddrCnt* and/or *ifAddrs* == NULL
- VOS_MEM_ERR** memory allocation error
- VOS SOCK_ERR** GetAdaptersInfo() error

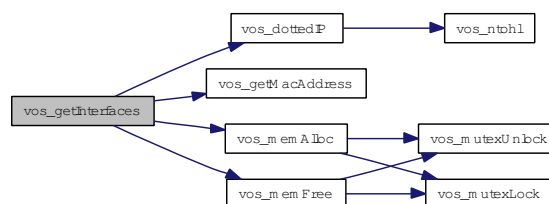
Parameters:

- ↔ *pAddrCnt* in: pointer to array size of interface record out: pointer to number of interface records read
- ↔ *ifAddrs* array of interface records

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** *pMAC* == NULL

Here is the call graph for this function:



5.31.3.3 EXT_DECL UINT32 vos_htonl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

- ← *val* Initial value.

Return values:

- swapped** value

5.31.3.4 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

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.31.3.5 EXT_DECL const CHAR8* vos_ipDotted (UINT32 *ipAddress*)

Convert IP address to dotted dec.

from !host! endianness

Parameters:

← *ipAddress* address in UINT32 in host endianness

Return values:

IP address as dotted decimal.

from !host! endianness.

Parameters:

← *ipAddress* address in UINT32 in host endianness

Return values:

IP address as dotted decimal.

5.31.3.6 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 a multicast address

FALSE address is not a multicast address

Parameters:

← *ipAddress* IP address to check.

Return values:

TRUE address is multicast

FALSE address is not a multicast address

5.31.3.7 EXT_DECL UINT32 vos_ntohl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.31.3.8 EXT_DECL UINT16 vos_ntohs (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.31.3.9 EXT_DECL INT32 vos_select (INT32 *highDesc*, VOS_FDS_T * *pReadableFD*, VOS_FDS_T * *pWriteableFD*, VOS_FDS_T * *pErrorFD*, VOS_TIME_T * *pTimeOut*)

select function.

Set the ready sockets in the supplied sets. Note: Some target systems might define this function as NOP.

Parameters:

- ← *highDesc* max. socket descriptor + 1
- ↔ *pReadableFD* pointer to readable socket set
- ↔ *pWritableFD* pointer to writeable socket set
- ↔ *pErrorFD* pointer to error socket set
- ← *pTimeOut* pointer to time out value

Return values:

number of ready file descriptors

5.31.3.10 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

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

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.

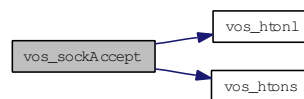
Parameters:

- ← *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:



5.31.3.11 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_PARAM_ERR** parameter out of range/invalid
- VOS_IO_ERR** Input/Output error
- VOS_MEM_ERR** resource error

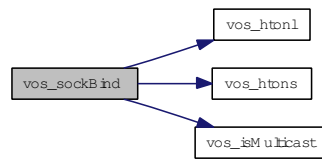
Parameters:

- ← *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:



5.31.3.12 EXT_DECL VOS_ERR_T vos_sockClose (INT32 *sock*)

Close a socket.

Release any resources acquired by this socket

Parameters:

← *sock* socket descriptor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pSock == NULL

Release any resources acquired by this socket

Parameters:

← *sock* socket descriptor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown

Release any resources acquired by this socket

Parameters:

← *sock* socket descriptor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown

5.31.3.13 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* parameter out of range/invalid
- VOS_IO_ERR* Input/Output 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

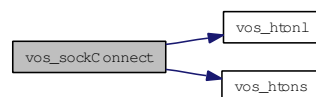
Parameters:

- ← *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:



5.31.3.14 EXT_DECL VOS_ERR_T vos_sockGetMAC (UINT8 *pMAC*[VOS_MAC_SIZE])

Return the MAC address of the default adapter.

Parameters:

- *pMAC* return MAC address.

Return values:*VOS_NO_ERR* no error*VOS_PARAM_ERR* pMAC == NULL*VOS SOCK_ERR* socket not available or option not supported

Here is the call graph for this function:

**5.31.3.15 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

Must be called once before any other call

Return values:*VOS_NO_ERR* no error*VOS SOCK_ERR* sockets not supported

Must be called once before any other call

Return values:*VOS_NO_ERR* no error*VOS SOCK_ERR* sockets not supported**5.31.3.16 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_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

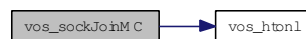
Return values:

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:



5.31.3.17 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

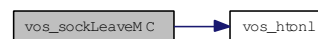
Return values:

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:



5.31.3.18 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_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

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

5.31.3.19 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

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

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:



5.31.3.20 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

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:



5.31.3.21 EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 *sock*, UINT8 * *pBuffer*, UINT32 * *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 occurred. 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
- VOS_BLOCK_ERR** call would have blocked in blocking mode

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 occurred. 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
- VOS_BLOCK_ERR** Call would have blocked in blocking mode

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 occurred. 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
- VOS_BLOCK_ERR** call would have blocked in blocking mode

5.31.3.22 EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 *sock*, UINT8 **pBuffer*, UINT32 **pSize*, UINT32 **pSrcIPAddr*, UINT16 **pSrcIPPort*, UINT32 **pDstIPAddr*, BOOL *peek*)

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 occurred. If called in non-blocking mode, and no data is available, VOS_NODATA_ERR will be returned. If pointers are provided, source IP, source port and destination IP will be reported on return.

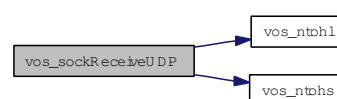
Parameters:

- ← *sock* socket descriptor
- *pBuffer* pointer to applications data buffer
- ↔ *pSize* pointer to the received data size
- *pSrcIPAddr* pointer to source IP
- *pSrcIPPort* pointer to source port
- *pDstIPAddr* pointer to dest IP
- ← *peek* if true, leave data in queue

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
- VOS_BLOCK_ERR** Call would have blocked in blocking mode

Here is the call graph for this function:



5.31.3.23 EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 * *pSize*)

Send TCP data.

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ↔ *pSize* In: size of the data to send, Out: no of bytes sent

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be sent
- VOS_NOCONN_ERR** no TCP connection
- VOS_BLOCK_ERR** call would have blocked in blocking mode, data partially sent

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ↔ *pSize* In: size of the data to send, Out: no of bytes sent

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be sent
- VOS_NOCONN_ERR** no TCP connection
- VOS_BLOCK_ERR** Call would have blocked in blocking mode

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ↔ *pSize* IN: bytes to send, OUT: bytes sent

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be sent
- VOS_NOCONN_ERR** no TCP connection
- VOS_BLOCK_ERR** Call would have blocked in blocking mode

5.31.3.24 EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 * *pSize*, 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
- ↔ *pSize* In: size of the data to send, Out: no of bytes sent
- ← *ipAddress* destination IP
- ← *port* destination port

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** parameter out of range/invalid
- VOS_IO_ERR** data could not be sent
- VOS_BLOCK_ERR** Call would have blocked in blocking mode

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ↔ *pSize* In: size of the data to send, Out: no of bytes sent
- ← *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_BLOCK_ERR** Call would have blocked in blocking mode

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ↔ *pSize* IN: bytes to send, OUT: bytes sent
- ← *ipAddress* destination IP
- ← *port* destination port

Return values:

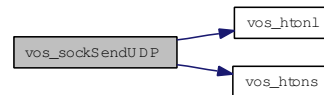
- VOS_NO_ERR** no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be sent

VOS_BLOCK_ERR Call would have blocked in blocking mode

Here is the call graph for this function:



5.31.3.25 EXT_DECL VOS_ERR_T vos_sockSetMulticastIf (INT32 sock, UINT32 mcIfAddress)

Set Using Multicast I/F.

Parameters:

← *sock* socket descriptor

← *mcIfAddress* using Multicast I/F Address

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

Parameters:

← *sock* socket descriptor

← *mcIfAddress* using Multicast I/F Address

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS SOCK_ERR option not supported

Parameters:

← *sock* socket descriptor

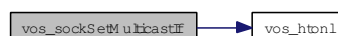
← *mcIfAddress* using Multicast I/F Address

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

Here is the call graph for this function:



5.31.3.26 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_PARAM_ERR* parameter out of range/invalid

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

Note: Some targeted systems might not support every option.

Parameters:

- ← *sock* socket descriptor
- ← *pOptions* pointer to socket options (optional)

Return values:

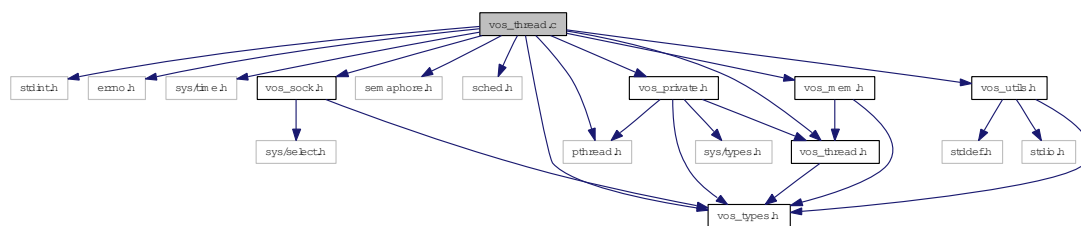
- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown

5.32 vos_thread.c File Reference

Multitasking functions.

```
#include <stdint.h>
#include <errno.h>
#include <sys/time.h>
#include <pthread.h>
#include <semaphore.h>
#include <sched.h>
#include "vos_sock.h"
#include "vos_types.h"
#include "vos_thread.h"
#include "vos_mem.h"
#include "vos_utils.h"
#include "vos_private.h"
```

Include dependency graph for posix/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 void [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 void [vos_clearTime](#) ([VOS_TIME_T](#) *pTime)
Clear the time stamp.
- EXT_DECL void [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 void [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 void [vos_divTime](#) ([VOS_TIME_T](#) *pTime, UINT32 divisor)
Divide the first time value by the second, return quotient in first.
- EXT_DECL void [vos_mulTime](#) ([VOS_TIME_T](#) *pTime, UINT32 mul)
Multiply the first time by the second, return product 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 void [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_mutexLocalCreate](#) (struct [VOS_MUTEX](#) *pMutex)
Create a recursive mutex.
- EXT_DECL void [vos_mutexDelete](#) ([VOS_MUTEX_T](#) pMutex)
Delete a mutex.
- EXT_DECL void [vos_mutexLocalDelete](#) (struct [VOS_MUTEX](#) *pMutex)
Delete a mutex.
- EXT_DECL [VOS_ERR_T vos_mutexLock](#) ([VOS_MUTEX_T](#) pMutex)
Take a mutex.
- EXT_DECL [VOS_ERR_T vos_mutexTryLock](#) ([VOS_MUTEX_T](#) pMutex)
Try to take a mutex.
- EXT_DECL [VOS_ERR_T vos_mutexUnlock](#) ([VOS_MUTEX_T](#) pMutex)

Release a mutex.

- EXT_DECL [VOS_ERR_T vos_semaCreate](#) ([VOS_SEMA_T](#) *pSema, [VOS_SEMA_STATE_T](#) initialState)

Create a semaphore.

- EXT_DECL void [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 void [vos_semaGive](#) ([VOS_SEMA_T](#) sema)

Give a semaphore.

5.32.1 Detailed Description

Multitasking functions.

OS abstraction of thread-handling functions

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

vos_thread.c 951 2013-06-13 13:56:42Z 97025

5.32.2 Function Documentation

5.32.2.1 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:



5.32.2.2 EXT_DECL void 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

5.32.2.3 EXT_DECL void vos_clearTime (VOS_TIME_T * *pTime*)

Clear the time stamp.

Parameters:

→ *pTime* Pointer to time value

5.32.2.4 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*

5.32.2.5 EXT_DECL void vos_divTime (VOS_TIME_T * *pTime*, UINT32 *divisor*)

Divide the first time value by the second, return quotient in first.

Divide the first time by the second, return quotient in first.

Parameters:

↔ *pTime* Pointer to time value

← *divisor* Divisor

5.32.2.6 EXT_DECL void vos_getTime (VOS_TIME_T * *pTime*)

Return the current time in sec and us.

Parameters:

→ *pTime* Pointer to time value

5.32.2.7 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"

5.32.2.8 EXT_DECL void 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

Here is the call graph for this function:

**5.32.2.9 EXT_DECL void vos_mulTime (VOS_TIME_T * *pTime*, UINT32 *mul*)**

Multiply the first time by the second, return product in first.

Parameters:

↔ *pTime* Pointer to time value

← *mul* Factor

5.32.2.10 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:

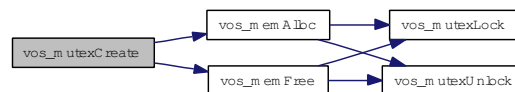
VOS_NO_ERR no error

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:



5.32.2.11 EXT_DECL void vos_mutexDelete (VOS_MUTEX_T pMutex)

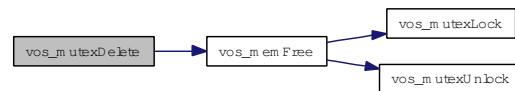
Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* mutex handle

Here is the call graph for this function:



5.32.2.12 EXT_DECL VOS_ERR_T vos_mutexLocalCreate (struct VOS_MUTEX * pMutex)

Create a recursive mutex.

Fill in a mutex handle. The mutex storage must be already allocated.

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

5.32.2.13 EXT_DECL void vos_mutexLocalDelete (struct VOS_MUTEX * *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* Pointer to mutex struct

5.32.2.14 EXT_DECL VOS_ERR_T vos_mutexLock (VOS_MUTEX_T *pMutex*)

Take a mutex.

Wait for the mutex to become available (lock).

Parameters:

← *pMutex* mutex handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR no such mutex

5.32.2.15 EXT_DECL VOS_ERR_T vos_mutexTryLock (VOS_MUTEX_T *pMutex*)

Try to take a mutex.

If mutex is can't be taken *VOS_MUTEX_ERR* is returned.

Parameters:

← *pMutex* mutex handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR mutex not locked

5.32.2.16 EXT_DECL VOS_ERR_T vos_mutexUnlock (VOS_MUTEX_T *pMutex*)

Release a mutex.

Unlock the mutex.

Parameters:

← *pMutex* mutex handle

5.32.2.17 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.

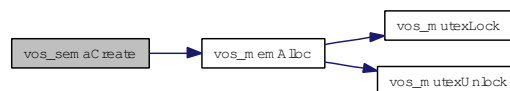
Parameters:

- *pSema* Pointer to semaphore handle
- ← *initialState* The initial state of the semaphore

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

Here is the call graph for this function:



5.32.2.18 EXT_DECL void vos_semaDelete (VOS_SEMA_T *sema*)

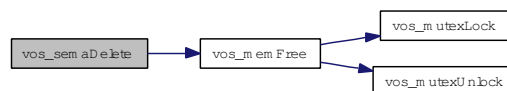
Delete a semaphore.

This will eventually release any processes waiting for the semaphore.

Parameters:

- ← *sema* semaphore handle

Here is the call graph for this function:



5.32.2.19 EXT_DECL void vos_semaGive (VOS_SEMA_T *sema*)

Give a semaphore.

Release (increase) a semaphore.

Parameters:

- ← *sema* semaphore handle

5.32.2.20 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 no wait

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

Here is the call graph for this function:

**5.32.2.21 EXT_DECL void 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

5.32.2.22 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

5.32.2.23 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

5.32.2.24 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

5.32.2.25 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

5.32.2.26 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

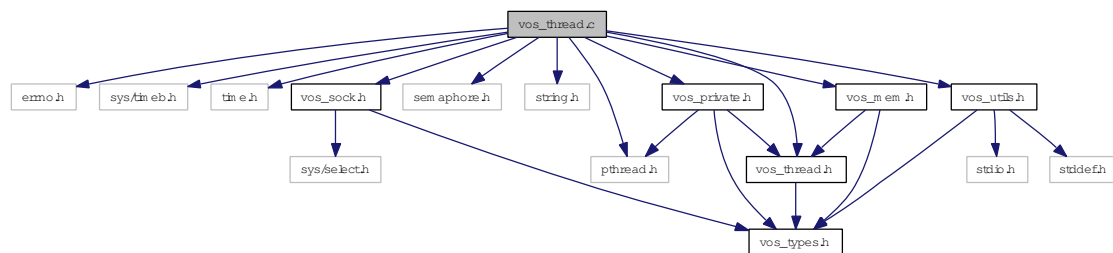
VOS_THREAD_ERR cancel failed

5.33 vos_thread.c File Reference

Multitasking functions.

```
#include <errno.h>
#include <sys/timeb.h>
#include <time.h>
#include <pthread.h>
#include <semaphore.h>
#include <string.h>
#include "vos_thread.h"
#include "vos_sock.h"
#include "vos_mem.h"
#include "vos_utils.h"
#include "vos_private.h"
```

Include dependency graph for windows/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.
- pthread_t * [vos_getFreeThreadHandle](#) (void)
Search a free Handle place in the thread handle list.
- 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 void [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 void [vos_clearTime](#) (VOS_TIME_T *pTime)
Clear the time stamp.
- EXT_DECL void [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 void [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 void [vos_divTime](#) (VOS_TIME_T *pTime, UINT32 divisor)
Divide the first time value by the second, return quotient in first.
- EXT_DECL void [vos_mulTime](#) (VOS_TIME_T *pTime, UINT32 mul)
Multiply the first time by the second, return product 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 void [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.
- [VOS_ERR_T vos_mutexLocalCreate](#) (struct VOS_MUTEX *pMutex)
Create a recursive mutex.
- EXT_DECL void [vos_mutexDelete](#) (VOS_MUTEX_T pMutex)
Delete a mutex.
- void [vos_mutexLocalDelete](#) (struct VOS_MUTEX *pMutex)
Delete a mutex.
- EXT_DECL [VOS_ERR_T vos_mutexLock](#) (VOS_MUTEX_T pMutex)
Take a mutex.
- EXT_DECL [VOS_ERR_T vos_mutexTryLock](#) (VOS_MUTEX_T pMutex)
Try to take a mutex.

- EXT_DECL [VOS_ERR_T](#) [vos_mutexUnlock](#) ([VOS_MUTEX_T](#) pMutex)
Release a mutex.
- EXT_DECL [VOS_ERR_T](#) [vos_semaCreate](#) ([VOS_SEMA_T](#) *pSema, [VOS_SEMA_STATE_T](#) initialState)
Create a semaphore.
- EXT_DECL void [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 void [vos_semaGive](#) ([VOS_SEMA_T](#) sema)
Give a semaphore.

5.33.1 Detailed Description

Multitasking functions.

OS abstraction of thread-handling functions

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2013. [vos_thread.c](#) uses pthreads-w32 (<http://sourceware.org/pthreads-win32/>) under LGPL license

Id

[vos_thread.c](#) 951 2013-06-13 13:56:42Z 97025

5.33.2 Function Documentation

5.33.2.1 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:



5.33.2.2 EXT_DECL void 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

5.33.2.3 EXT_DECL void vos_clearTime (VOS_TIME_T * *pTime*)

Clear the time stamp.

Parameters:

- *pTime* Pointer to time value

5.33.2.4 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*

5.33.2.5 EXT_DECL void vos_divTime (VOS_TIME_T * *pTime*, UINT32 *divisor*)

Divide the first time value by the second, return quotient in first.

Divide the first time by the second, return quotient in first.

Parameters:

↔ *pTime* Pointer to time value

← *divisor* Divisor

5.33.2.6 pthread_t* vos_getFreeThreadHandle (void)

Search a free Handle place in the thread handle list.

Return values:

pointer to a free thread handle or NULL if not available

5.33.2.7 EXT_DECL void vos_getTime (VOS_TIME_T * *pTime*)

Return the current time in sec and us.

Parameters:

→ *pTime* Pointer to time value

5.33.2.8 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"

5.33.2.9 EXT_DECL void 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

Here is the call graph for this function:



5.33.2.10 EXT_DECL void vos_mulTime (VOS_TIME_T * *pTime*, UINT32 *mul*)

Multiply the first time by the second, return product in first.

Parameters:

↔ *pTime* Pointer to time value

← *mul* Factor

5.33.2.11 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:

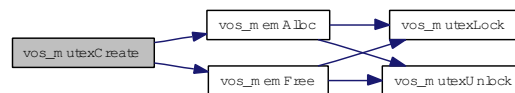
VOS_NO_ERR no error

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:

**5.33.2.12 EXT_DECL void vos_mutexDelete (VOS_MUTEX_T *pMutex*)**

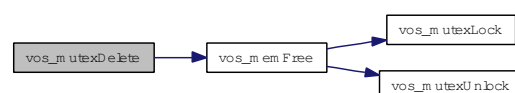
Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* mutex handle

Here is the call graph for this function:



5.33.2.13 VOS_ERR_T vos_mutexLocalCreate (struct VOS_MUTEX * *pMutex*)

Create a recursive mutex.

Fill in a mutex handle. The mutex storage must be already allocated.

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

5.33.2.14 void vos_mutexLocalDelete (struct VOS_MUTEX * *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* Pointer to mutex struct

5.33.2.15 EXT_DECL VOS_ERR_T vos_mutexLock (VOS_MUTEX_T *pMutex*)

Take a mutex.

Wait for the mutex to become available (lock).

Parameters:

← *pMutex* mutex handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR no such mutex

5.33.2.16 EXT_DECL VOS_ERR_T vos_mutexTryLock (VOS_MUTEX_T *pMutex*)

Try to take a mutex.

If mutex is can't be taken *VOS_MUTEX_ERR* is returned.

Parameters:

← *pMutex* mutex handle

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR pMutex == NULL or wrong type
VOS_MUTEX_ERR mutex not locked

5.33.2.17 EXT_DECL VOS_ERR_T vos_mutexUnlock (VOS_MUTEX_T pMutex)

Release a mutex.

Unlock the mutex.

Parameters:

← **pMutex** mutex handle

5.33.2.18 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.

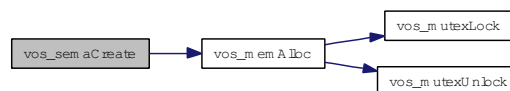
Parameters:

→ **pSema** Pointer to semaphore handle
 ← **initialState** The initial state of the semaphore

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

Here is the call graph for this function:

**5.33.2.19 EXT_DECL void vos_semaDelete (VOS_SEMA_T sema)**

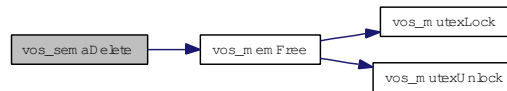
Delete a semaphore.

This will eventually release any processes waiting for the semaphore.

Parameters:

← **sema** semaphore handle

Here is the call graph for this function:



5.33.2.20 EXT_DECL void vos_semaGive (VOS_SEMA_T *sema*)

Give a semaphore.

Release (increase) a semaphore.

Parameters:

← *sema* semaphore handle

5.33.2.21 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 no wait

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

Here is the call graph for this function:



5.33.2.22 EXT_DECL void 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

5.33.2.23 `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

Here is the call graph for this function:



5.33.2.24 `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

5.33.2.25 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

5.33.2.26 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

5.33.2.27 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

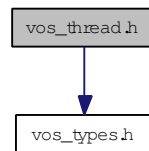
VOS_THREAD_ERR cancel failed

5.34 vos_thread.h File Reference

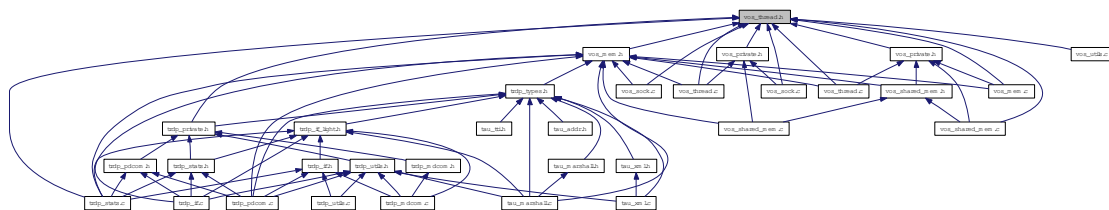
Threading functions for OS abstraction.

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

Include dependency graph for vos_thread.h:



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



Defines

- #define **VOS_MAX_THREAD_CNT** 100
The maximum number of concurrent usable threads.
- #define **VOS_SEMA_WAIT_FOREVER** 0xFFFFFFFF
Timeout value to wait forever for a semaphore.

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 * **VOS_MUTEX_T**
Hidden mutex handle definition.
- typedef struct VOS_SEMA * **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 void [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 void [vos_clearTime](#) ([VOS_TIME_T](#) *pTime)
Clear the time stamp.
- EXT_DECL void [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 void [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 void [vos_divTime](#) ([VOS_TIME_T](#) *pTime, [UINT32](#) divisor)
Divide the first time by the second, return quotient in first.

- EXT_DECL void [vos_mulTime](#) (VOS_TIME_T *pTime, UINT32 mul)
Multiply the first time by the second, return product in first.
- EXT_DECL void [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 void [vos_mutexDelete](#) (VOS_MUTEX_T pMutex)
Delete a mutex.
- EXT_DECL [VOS_ERR_T](#) [vos_mutexLock](#) (VOS_MUTEX_T pMutex)
Take a mutex.
- EXT_DECL [VOS_ERR_T](#) [vos_mutexTryLock](#) (VOS_MUTEX_T pMutex)
Try to take a mutex.
- EXT_DECL [VOS_ERR_T](#) [vos_mutexUnlock](#) (VOS_MUTEX_T pMutex)
Release a mutex.
- EXT_DECL [VOS_ERR_T](#) [vos_semaCreate](#) (VOS_SEMA_T *pSema, [VOS_SEMA_STATE_T](#) initialState)
Create a semaphore.
- EXT_DECL void [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 void [vos_semaGive](#) (VOS_SEMA_T sema)
Give a semaphore.

5.34.1 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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[vos_thread.h](#) 951 2013-06-13 13:56:42Z 97025

5.34.2 Function Documentation

5.34.2.1 EXT_DECL void 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
- ↔ *pTime* Pointer to time value
- ← *pAdd* Pointer to time value

5.34.2.2 EXT_DECL void vos_clearTime (VOS_TIME_T * *pTime*)

Clear the time stamp.

Parameters:

- *pTime* Pointer to time value
- *pTime* Pointer to time value

5.34.2.3 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*

Parameters:

- ↔ *pTime* Pointer to time value
- ← *pCmp* Pointer to time value to compare

Return values:

- 0 *pTime* == *pCmp*
- 1 *pTime* < *pCmp*
- 1 *pTime* > *pCmp*

5.34.2.4 EXT_DECL void vos_divTime (VOS_TIME_T * *pTime*, UINT32 *divisor*)

Divide the first time by the second, return quotient in first.

Parameters:

- ↔ *pTime* Pointer to time value
- ← *divisor* Divisor

Divide the first time by the second, return quotient in first.

Parameters:

- ↔ *pTime* Pointer to time value
- ← *divisor* Divisor

5.34.2.5 EXT_DECL void vos_getTime (VOS_TIME_T * *pTime*)

Return the current time in sec and us.

Parameters:

- *pTime* Pointer to time value
- *pTime* Pointer to time value

5.34.2.6 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"

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"

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"

5.34.2.7 EXT_DECL void 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
- *pUuid* Pointer to a universal unique identifier

Here is the call graph for this function:



5.34.2.8 EXT_DECL void vos_mulTime (VOS_TIME_T * *pTime*, UINT32 *mul*)

Multiply the first time by the second, return product in first.

Parameters:

- ↔ *pTime* Pointer to time value
- ← *mul* Factor

5.34.2.9 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:

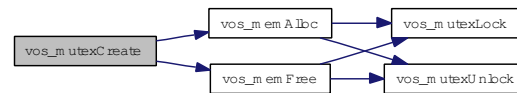
- VOS_NO_ERR* no error

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:



5.34.2.10 EXT_DECL void vos_mutexDelete (VOS_MUTEX_T pMutex)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← **pMutex** mutex handle

Return values:

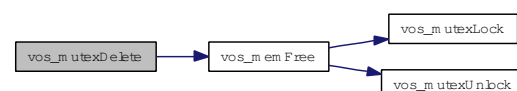
VOS_NO_ERR no error

Release the resources taken by the mutex.

Parameters:

← **pMutex** mutex handle

Here is the call graph for this function:



5.34.2.11 EXT_DECL VOS_ERR_T vos_mutexLock (VOS_MUTEX_T pMutex)

Take a mutex.

Wait for the mutex to become available (lock).

Parameters:

← **pMutex** 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:

← *pMutex* mutex handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMutex == NULL or wrong type

VOS_MUTEX_ERR no such mutex

5.34.2.12 EXT_DECL VOS_ERR_T vos_mutexTryLock (VOS_MUTEX_T pMutex)

Try to take a mutex.

If mutex is can't be taken VOS_MUTEX_ERR is returned.

Parameters:

← *pMutex* 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:

← *pMutex* mutex handle

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMutex == NULL or wrong type

VOS_MUTEX_ERR mutex not locked

5.34.2.13 EXT_DECL VOS_ERR_T vos_mutexUnlock (VOS_MUTEX_T pMutex)

Release a mutex.

Unlock the mutex.

Parameters:

← *pMutex* mutex handle

Unlock the mutex.

Parameters:

← *pMutex* mutex handle

Unlock the mutex.

Parameters:

← *pMutex* mutex handle

5.34.2.14 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 semaphore

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

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 semaphore

Return values:

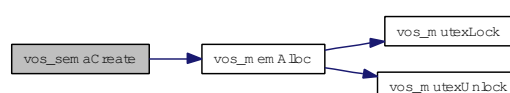
VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

VOS_SEMA_ERR no semaphore available

Here is the call graph for this function:



5.34.2.15 EXT_DECL void vos_semaDelete (VOS_SEMA_T *sema*)

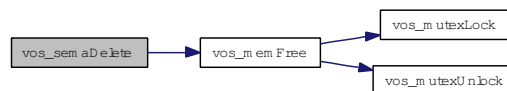
Delete a semaphore.

This will eventually release any processes waiting for the semaphore.

Parameters:

← *sema* semaphore handle

Here is the call graph for this function:



5.34.2.16 EXT_DECL void vos_semaGive (VOS_SEMA_T *sema*)

Give a semaphore.

Release (increase) a semaphore.

Parameters:

← *sema* semaphore handle

5.34.2.17 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 no wait

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

Try to get (decrease) a semaphore.

Parameters:

← *sema* semaphore handle

← *timeout* Max. time in us to wait, 0 means no wait

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

Here is the call graph for this function:



5.34.2.18 EXT_DECL void 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
 ↔ **pTime** Pointer to time value
 ← **pSub** Pointer to time value

5.34.2.19 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

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

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

Here is the call graph for this function:



5.34.2.20 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

5.34.2.21 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

Must be called once before any other call

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR threading not supported

Must be called once before any other call

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR threading not supported

5.34.2.22 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

5.34.2.23 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:

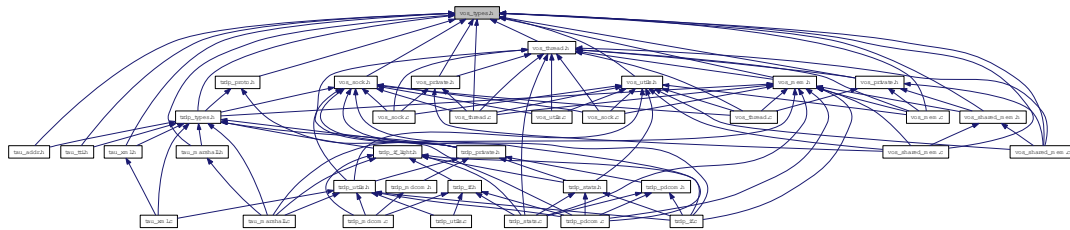
VOS_NO_ERR no error

VOS_THREAD_ERR cancel failed

5.35 vos_types.h File Reference

Typedefs for OS abstraction.

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



Data Structures

- struct [VOS_TIME_T](#)
Timer value compatible with timeval / select.

Defines

- #define [INLINE](#) inline
inline macros

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_SOCKET_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_BLOCK_ERR = -14,  
VOS_INTEGRATION_ERR = -15,  
VOS_NOCONN_ERR = -16,  
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.

5.35.1 Detailed Description

Typedefs for OS abstraction.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[vos_types.h](#) 951 2013-06-13 13:56:42Z 97025

5.35.2 Typedef Documentation

5.35.2.1 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

5.35.3 Enumeration Type Documentation

5.35.3.1 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** Timeout.
- 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_BLOCK_ERR** System call would have blocked in blocking mode.
- VOS_INTEGRATION_ERR** Alignment or endianness for selected target wrong.
- VOS_NOCONN_ERR** No TCP connection.
- VOS_UNKNOWN_ERR** Unknown error.

5.35.3.2 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.

5.35.4 Function Documentation

5.35.4.1 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

Initialize the vos library.

Parameters:

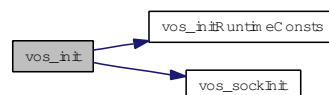
← *pRefCon* context for debug output function

← *pDebugOutput* Pointer to debug output function.

Return values:

VOS_NO_ERR no error **VOS_INTEGRATION_ERR** if endianness/alignment mismatch

Here is the call graph for this function:

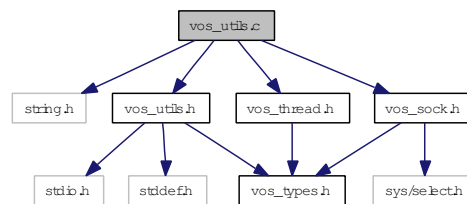


5.36 vos_utils.c File Reference

Common functions for VOS.

```
#include <string.h>
#include "vos_utils.h"
#include "vos_sock.h"
#include "vos_thread.h"
```

Include dependency graph for vos_utils.c:



Functions

- [VOS_ERR_T vos_initRuntimeConsts](#) (void)
Pre-compute alignment and endianness.
- [VOS_ERR_T vos_init](#) (void *pRefCon, [VOS_PRINT_DBG_T](#) pDebugOutput)
Initialize the virtual operating system.
- [UINT32 vos_crc32](#) (UINT32 crc, const [UINT8](#) *pData, [UINT32](#) dataLen)
Compute crc32 according to IEEE802.3.
- [INLINE BOOL vos_isBigEndian](#) (void)
Return endianness.

5.36.1 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:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[vos_utils.c](#) 951 2013-06-13 13:56:42Z 97025

5.36.2 Function Documentation**5.36.2.1** `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

5.36.2.2 `VOS_ERR_T vos_init (void * pRefCon, VOS_PRINT_DBG_T pDebugOutput)`

Initialize the virtual operating system.

Initialize the vos library.

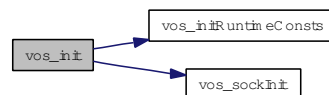
Parameters:

- ← *pRefCon* context for debug output function
- ← *pDebugOutput* Pointer to debug output function.

Return values:

VOS_NO_ERR no error *VOS_INTEGRATION_ERR* if endianness/alignment mismatch

Here is the call graph for this function:

**5.36.2.3** `VOS_ERR_T vos_initRuntimeConsts (void)`

Pre-compute alignment and endianness.

Return values:

VOS_INTEGRATION_ERR or *VOS_NO_ERR*

5.36.2.4 INLINE BOOL vos_isBigEndian (void)

Return endianness.

Return values:

TRUE if big endian

Functions

- EXT_DECL UINT32 [vos_crc32](#) (UINT32 crc, const UINT8 *pData, UINT32 dataLen)
Calculate CRC for the given buffer and length.

5.37.1 Detailed Description

Typedefs for OS abstraction.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, Copyright Bombardier Transportation GmbH, Germany, 2012.

Id

[vos_utils.h](#) 951 2013-06-13 13:56:42Z 97025

5.37.2 Define Documentation

5.37.2.1 #define VOS_MAX_ERR_STR_SIZE (VOS_MAX_PRNT_STR_SIZE - VOS_MAX_FRMT_SIZE)

Max.

size of the error part

5.37.2.2 #define VOS_MAX_FRMT_SIZE 64

Max.

size of the 'format' part

5.37.2.3 #define VOS_MAX_PRNT_STR_SIZE 256

String size definitions for the debug output functions.

Max. size of the debug/error string of debug function

5.37.3 Function Documentation

5.37.3.1 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

- am_big_endian
 - trdp_utils.c, [236](#)
 - trdp_utils.h, [246](#)
- cyclicThread
 - posix/vos_thread.c, [337](#)
 - windows/vos_thread.c, [348](#)
- datasetLength
 - GNU_PACKED, [10](#)
- destAddr
 - TRDP_PUB_STATISTICS_T, [50](#)
- filterAddr
 - TRDP_SUBS_STATISTICS_T, [61](#)
- GNU_PACKED, [9](#)
 - datasetLength, [10](#)
 - msgType, [10](#)
 - protocolVersion, [10](#)
- MD_ELE, [12](#)
 - pPacket, [14](#)
- MD_LIS_ELE, [15](#)
- msgType
 - GNU_PACKED, [10](#)
 - TRDP_MD_INFO_T, [37](#)
 - TRDP_PD_INFO_T, [45](#)
- numRecv
 - TRDP_SUBS_STATISTICS_T, [62](#)
- operator
 - TRDP_TRAIN_INFO_T, [65](#)
- orient
 - TRDP_CAR_INFO_T, [21](#)
 - TRDP_CST_INFO_T, [24](#)
 - TRDP_DEVICE_INFO_T, [29](#)
- owner
 - TRDP_CST_INFO_T, [24](#)
- pCarInfo
 - TRDP_CST_INFO_T, [24](#)
- pCstInfo
 - TRDP_TRAIN_INFO_T, [65](#)
- PD_ELE, [16](#)
- pFrame, [17](#)
- pDevInfo
 - TRDP_CAR_INFO_T, [21](#)
- pFctInfo
 - TRDP_CST_INFO_T, [24](#)
- pFrame
 - PD_ELE, [17](#)
- posix/vos_private.h
 - vos_mutexLocalCreate, [273](#)
 - vos_mutexLocalDelete, [273](#)
- posix/vos_shared_mem.c
 - vos_sharedClose, [277](#)
 - vos_sharedOpen, [277](#)
- posix/vos_sock.c
 - vos_dottedIP, [288](#)
 - vos_getInterfaces, [288](#)
 - vos_getMacAddress, [288](#)
 - vos_htonl, [288](#)
 - vos_htons, [289](#)
 - vos_ipDotted, [289](#)
 - vos_isMulticast, [289](#)
 - vos_ntohl, [289](#)
 - vos_ntohs, [290](#)
 - vos_select, [290](#)
 - vos_sockAccept, [290](#)
 - vos_sockBind, [291](#)
 - vos_sockClose, [291](#)
 - vos_sockConnect, [292](#)
 - vos_sockGetMAC, [292](#)
 - vos_sockInit, [293](#)
 - vos_sockJoinMC, [293](#)
 - vos_sockLeaveMC, [293](#)
 - vos_sockListen, [294](#)
 - vos_sockOpenTCP, [294](#)
 - vos_sockOpenUDP, [295](#)
 - vos_sockReceiveTCP, [295](#)
 - vos_sockReceiveUDP, [296](#)
 - vos_sockSendTCP, [296](#)
 - vos_sockSendUDP, [297](#)
 - vos_sockSetMulticastIf, [297](#)
 - vos_sockSetOptions, [298](#)
- posix/vos_thread.c
 - cyclicThread, [337](#)
 - vos_addTime, [338](#)
 - vos_clearTime, [338](#)

- vos_cmpTime, 338
- vos_divTime, 338
- vos_getTime, 338
- vos_getTimeStamp, 339
- vos_getUuid, 339
- vos_mulTime, 339
- vos_mutexCreate, 339
- vos_mutexDelete, 340
- vos_mutexLocalCreate, 340
- vos_mutexLocalDelete, 340
- vos_mutexLock, 341
- vos_mutexTryLock, 341
- vos_mutexUnlock, 341
- vos_semaCreate, 341
- vos_semaDelete, 342
- vos_semaGive, 342
- vos_semaTake, 342
- vos_subTime, 343
- vos_threadCreate, 343
- vos_threadDelay, 344
- vos_threadInit, 344
- vos_threadIsActive, 344
- vos_threadTerminate, 344
- pPacket
 - MD_ELE, 14
- protocolVersion
 - GNU_PACKED, 10
- qos
 - VOS_SOCK_OPT_T, 68
- tau_tti.h
 - TRDP_FCT_CAR, 94
 - TRDP_FCT_CST, 95
 - TRDP_FCT_INVALID, 94
 - TRDP_FCT_TRAIN, 95
 - TRDP_INAUG_INVALID, 95
 - TRDP_INAUG_LEAD_CONF, 95
 - TRDP_INAUG_LEAD_UNCONF, 95
 - TRDP_INAUG_NOLEAD_UNCONF, 95
- tau_xml.h
 - TRDP_DBG_CAT, 107
 - TRDP_DBG_DBG, 107
 - TRDP_DBG_DEFAULT, 107
 - TRDP_DBG_ERR, 107
 - TRDP_DBG_INFO, 107
 - TRDP_DBG_LOC, 107
 - TRDP_DBG_OFF, 107
 - TRDP_DBG_TIME, 107
 - TRDP_DBG_WARN, 107
- tau_addr.h, 71
 - tau_addr2CarId, 73
 - tau_addr2CarNo, 74
 - tau_addr2CstId, 74
 - tau_addr2CstNo, 74
 - tau_addr2IecCarNo, 75
 - tau_addr2IecCstNo, 75
 - tau_addr2Uri, 75
 - tau_carNo2Ids, 76
 - tau_cstNo2CstId, 76
 - tau_getOwnAddr, 76
 - tau_getOwnIds, 76
 - tau_iecCarNo2Ids, 77
 - tau_iecCstNo2CstId, 77
 - tau_label2CarId, 77
 - tau_label2CarNo, 78
 - tau_label2CstId, 78
 - tau_label2CstNo, 78
 - tau_label2IecCarNo, 79
 - tau_label2IecCstNo, 79
 - tau_uri2Addr, 79
- tau_addr2CarId
 - tau_addr.h, 73
- tau_addr2CarNo
 - tau_addr.h, 74
- tau_addr2CstId
 - tau_addr.h, 74
- tau_addr2CstNo
 - tau_addr.h, 74
- tau_addr2IecCarNo
 - tau_addr.h, 75
- tau_addr2IecCstNo
 - tau_addr.h, 75
- tau_addr2Uri
 - tau_addr.h, 75
- tau_calcDatasetSize
 - tau_marshall.c, 82
 - tau_marshall.h, 87
- tau_calcDatasetSizeByComId
 - tau_marshall.c, 83
 - tau_marshall.h, 88
- tau_carNo2Ids
 - tau_addr.h, 76
- tau_cstNo2CstId
 - tau_addr.h, 76
- tau_freeTelegrams
 - tau_xml.c, 102
 - tau_xml.h, 107
- tau_freeXmlDoc
 - tau_xml.c, 102
 - tau_xml.h, 107
- tau_getCarDevCnt
 - tau_tti.h, 95
- tau_getCarInfo
 - tau_tti.h, 95
- tau_getCarOrient
 - tau_tti.h, 96
- tau_getCstCarCnt

- tau_tti.h, 96
- tau_getCstFctCnt
 - tau_tti.h, 96
- tau_getCstFctInfo
 - tau_tti.h, 97
- tau_getCstInfo
 - tau_tti.h, 97
- tau_getDevInfo
 - tau_tti.h, 97
- tau_getEtbState
 - tau_tti.h, 98
- tau_getIecCarOrient
 - tau_tti.h, 98
- tau_getOwnAddr
 - tau_addr.h, 76
- tau_getOwnIds
 - tau_addr.h, 76
- tau_getTrnCarCnt
 - tau_tti.h, 99
- tau_getTrnCstCnt
 - tau_tti.h, 99
- tau_getTrnInfo
 - tau_tti.h, 99
- tau_iecCarNo2Ids
 - tau_addr.h, 77
- tau_iecCstNo2CstId
 - tau_addr.h, 77
- tau_initMarshall
 - tau_marshall.c, 83
 - tau_marshall.h, 88
- tau_label2CarId
 - tau_addr.h, 77
- tau_label2CarNo
 - tau_addr.h, 78
- tau_label2CstId
 - tau_addr.h, 78
- tau_label2CstNo
 - tau_addr.h, 78
- tau_label2IecCarNo
 - tau_addr.h, 79
- tau_label2IecCstNo
 - tau_addr.h, 79
- tau_marshall
 - tau_marshall.c, 84
 - tau_marshall.h, 89
- tau_marshall.c, 81
 - tau_calcDatasetSize, 82
 - tau_calcDatasetSizeByComId, 83
 - tau_initMarshall, 83
 - tau_marshall, 84
 - tau_marshallIds, 84
 - tau_unmarshall, 85
 - tau_unmarshallIds, 85
- tau_marshall.h, 86
 - tau_calcDatasetSize, 87
 - tau_calcDatasetSizeByComId, 88
 - tau_initMarshall, 88
 - tau_marshall, 89
 - tau_marshallIds, 89
 - tau_unmarshall, 90
 - tau_unmarshallIds, 90
- TAU_MARSHALL_INFO_T, 19
- tau_marshallDs
 - tau_marshall.c, 84
 - tau_marshall.h, 89
- tau_prepareXmlDoc
 - tau_xml.c, 102
 - tau_xml.h, 108
- tau_readXmlDatasetConfig
 - tau_xml.c, 102
 - tau_xml.h, 108
- tau_readXmlDeviceConfig
 - tau_xml.c, 103
 - tau_xml.h, 108
- tau_readXmlInterfaceConfig
 - tau_xml.c, 103
 - tau_xml.h, 109
- tau_tti.h, 92
 - tau_getCarDevCnt, 95
 - tau_getCarInfo, 95
 - tau_getCarOrient, 96
 - tau_getCstCarCnt, 96
 - tau_getCstFctCnt, 96
 - tau_getCstFctInfo, 97
 - tau_getCstInfo, 97
 - tau_getDevInfo, 97
 - tau_getEtbState, 98
 - tau_getIecCarOrient, 98
 - tau_getTrnCarCnt, 99
 - tau_getTrnCstCnt, 99
 - tau_getTrnInfo, 99
- TRDP_FCT_T, 94
- TRDP_INAUG_STATE_T, 95
- tau_unmarshall
 - tau_marshall.c, 85
 - tau_marshall.h, 90
- tau_unmarshallDs
 - tau_marshall.c, 85
 - tau_marshall.h, 90
- tau_uri2Addr
 - tau_addr.h, 79
- tau_xml.c, 100
 - tau_freeTelegrams, 102
 - tau_freeXmlDoc, 102
 - tau_prepareXmlDoc, 102
 - tau_readXmlDatasetConfig, 102
 - tau_readXmlDeviceConfig, 103
 - tau_readXmlInterfaceConfig, 103

- TRDP_SDT_DEFAULT_CMTHR, 102
- tau_xml.h, 105
 - tau_freeTelegrams, 107
 - tau_freeXmlDoc, 107
 - tau_prepareXmlDoc, 108
 - tau_readXmlDatasetConfig, 108
 - tau_readXmlDeviceConfig, 108
 - tau_readXmlInterfaceConfig, 109
 - TRDP_DBG_OPTION_T, 107
- timeout
 - TRDP_SUBS_STATISTICS_T, 61
- tlc_closeSession
 - trdp_if.c, 115
 - trdp_if_light.h, 139
- tlc_freeBuf
 - trdp_if_light.h, 140
- tlc_getInterval
 - trdp_if.c, 115
 - trdp_if_light.h, 140
- tlc_getJoinStatistics
 - trdp_if_light.h, 141
 - trdp_stats.c, 216
- tlc_getListStatistics
 - trdp_if_light.h, 142
 - trdp_stats.c, 217
- tlc_getPubStatistics
 - trdp_if_light.h, 143
 - trdp_stats.c, 217
- tlc_getRedStatistics
 - trdp_if_light.h, 144
 - trdp_stats.c, 218
- tlc_getStatistics
 - trdp_if_light.h, 144
 - trdp_stats.c, 218
- tlc_getSubsStatistics
 - trdp_if_light.h, 145
 - trdp_stats.c, 219
- tlc_getVersion
 - trdp_if.c, 116
 - trdp_if_light.h, 146
- tlc_getVersionString
 - trdp_if.c, 116
 - trdp_if_light.h, 146
- tlc_init
 - trdp_if.c, 116
 - trdp_if_light.h, 147
- tlc_openSession
 - trdp_if.c, 117
 - trdp_if_light.h, 147
- tlc_process
 - trdp_if.c, 119
 - trdp_if_light.h, 150
- tlc_reinitSession
 - trdp_if.c, 120
- trdp_if_light.h, 151
- tlc_resetStatistics
 - trdp_if_light.h, 152
 - trdp_stats.c, 220
- tlc_setTopoCount
 - trdp_if.c, 121
 - trdp_if_light.h, 153
- tlc_terminate
 - trdp_if.c, 121
 - trdp_if_light.h, 153
- tlm_abortSession
 - trdp_if_light.h, 154
- tlm_addListener
 - trdp_if_light.h, 154
- tlm_confirm
 - trdp_if_light.h, 155
- tlm_delListener
 - trdp_if_light.h, 156
- tlm_notify
 - trdp_if_light.h, 156
- tlm_reply
 - trdp_if_light.h, 157
- tlm_replyErr
 - trdp_if_light.h, 157
- tlm_replyQuery
 - trdp_if_light.h, 158
- tlm_request
 - trdp_if_light.h, 159
- tlp_get
 - trdp_if.c, 122
 - trdp_if_light.h, 160
- tlp_getRedundant
 - trdp_if.c, 123
 - trdp_if_light.h, 161
- tlp_publish
 - trdp_if.c, 124
 - trdp_if_light.h, 162
- tlp_put
 - trdp_if.c, 126
 - trdp_if_light.h, 164
- tlp_request
 - trdp_if.c, 127
 - trdp_if_light.h, 165
- tlp_setRedundant
 - trdp_if.c, 128
 - trdp_if_light.h, 167
- tlp_subscribe
 - trdp_if.c, 129
 - trdp_if_light.h, 168
- tlp_unpublish
 - trdp_if.c, 130
 - trdp_if_light.h, 170
- tlp_unsubscribe
 - trdp_if.c, 131

- trdp_if_light.h, 171
- toBehav
 - TRDP_SUBS_STATISTICS_T, 61
- topoCnt
 - TRDP_TRAIN_INFO_T, 65
- TRDP_APP_CONFIRMTO_ERR
 - trdp_types.h, 232
- TRDP_APP_REPLYTO_ERR
 - trdp_types.h, 232
- TRDP_APP_TIMEOUT_ERR
 - trdp_types.h, 232
- TRDP_BLOCK_ERR
 - trdp_types.h, 232
- TRDP_BOOLEAN
 - trdp_types.h, 231
- TRDP_CHAR8
 - trdp_types.h, 231
- TRDP_COMID_ERR
 - trdp_types.h, 232
- TRDP_CONFIRMTO_ERR
 - trdp_types.h, 232
- TRDP_CRC_ERR
 - trdp_types.h, 232
- TRDP_DBG_CAT
 - tau_xml.h, 107
- TRDP_DBG_DBG
 - tau_xml.h, 107
- TRDP_DBG_DEFAULT
 - tau_xml.h, 107
- TRDP_DBG_ERR
 - tau_xml.h, 107
- TRDP_DBG_INFO
 - tau_xml.h, 107
- TRDP_DBG_LOC
 - tau_xml.h, 107
- TRDP_DBG_OFF
 - tau_xml.h, 107
- TRDP_DBG_TIME
 - tau_xml.h, 107
- TRDP_DBG_WARN
 - tau_xml.h, 107
- TRDP_FCT_CAR
 - tau_tti.h, 94
- TRDP_FCT_CST
 - tau_tti.h, 95
- TRDP_FCT_INVALID
 - tau_tti.h, 94
- TRDP_FCT_TRAIN
 - tau_tti.h, 95
- TRDP_FLAGS_CALLBACK
 - trdp_types.h, 232
- TRDP_FLAGS_DEFAULT
 - trdp_types.h, 232
- TRDP_FLAGS_MARSHALL
 - trdp_types.h, 232
- TRDP_FLAGS_NONE
 - trdp_types.h, 232
- TRDP_FLAGS_TCP
 - trdp_types.h, 232
- TRDP_INAUG_INVALID
 - tau_tti.h, 95
- TRDP_INAUG_LEAD_CONF
 - tau_tti.h, 95
- TRDP_INAUG_LEAD_UNCONF
 - tau_tti.h, 95
- TRDP_INAUG_NOLEAD_UNCONF
 - tau_tti.h, 95
- TRDP_INIT_ERR
 - trdp_types.h, 232
- TRDP_INT16
 - trdp_types.h, 231
- TRDP_INT32
 - trdp_types.h, 231
- TRDP_INT64
 - trdp_types.h, 231
- TRDP_INT8
 - trdp_types.h, 231
- TRDP_INTEGRATION_ERR
 - trdp_types.h, 232
- TRDP_INVALID_DATA
 - trdp_private.h, 210
- TRDP_IO_ERR
 - trdp_types.h, 232
- TRDP_MEM_ERR
 - trdp_types.h, 232
- TRDP_MSG_MC
 - trdp_proto.h, 214
- TRDP_MSG_ME
 - trdp_proto.h, 214
- TRDP_MSG_MN
 - trdp_proto.h, 214
- TRDP_MSG_MP
 - trdp_proto.h, 214
- TRDP_MSG_MQ
 - trdp_proto.h, 214
- TRDP_MSG_MR
 - trdp_proto.h, 214
- TRDP_MSG_PD
 - trdp_proto.h, 214
- TRDP_MSG_PE
 - trdp_proto.h, 214
- TRDP_MSG_PP
 - trdp_proto.h, 214
- TRDP_MSG_PR
 - trdp_proto.h, 214
- TRDP_MUTEX_ERR
 - trdp_types.h, 232
- TRDP_NO_ERR

- trdp_types.h, 231
- TRDP_NOCONN_ERR
 - trdp_types.h, 232
- TRDP_NODATA_ERR
 - trdp_types.h, 232
- TRDP_NOINIT_ERR
 - trdp_types.h, 232
- TRDP_NOLIST_ERR
 - trdp_types.h, 232
- TRDP_NOPUB_ERR
 - trdp_types.h, 232
- TRDP_NOSESSION_ERR
 - trdp_types.h, 232
- TRDP_NOSUB_ERR
 - trdp_types.h, 232
- TRDP_OPTION_BLOCK
 - trdp_types.h, 233
- TRDP_OPTION_TRAFFIC_SHAPING
 - trdp_types.h, 233
- TRDP_PACKET_ERR
 - trdp_types.h, 232
- TRDP_PARAM_ERR
 - trdp_types.h, 231
- trdp_private.h
 - TRDP_INVALID_DATA, 210
 - TRDP_PULL_SUB, 210
 - TRDP_REDUNDANT, 210
 - TRDP_REQ_2B_SENT, 210
 - TRDP SOCK_MD_TCP, 210
 - TRDP SOCK_MD_UDP, 210
 - TRDP SOCK_PD, 210
 - TRDP_ST_NONE, 209
 - TRDP_ST_RX_CONF_RECEIVED, 210
 - TRDP_ST_RX_NOTIFY_RECEIVED, 210
 - TRDP_ST_RX_READY, 210
 - TRDP_ST_RX_REPLY_SENT, 210
 - TRDP_ST_RX_REPLYQUERY_W4C, 210
 - TRDP_ST_RX_REQ_W4AP_REPLY, 210
 - TRDP_ST_TX_CONFIRM_ARM, 210
 - TRDP_ST_TX_NOTIFY_ARM, 209
 - TRDP_ST_TX_REPLY_ARM, 209
 - TRDP_ST_TX_REPLY_RECEIVED, 210
 - TRDP_ST_TX_REPLYQUERY_ARM, 209
 - TRDP_ST_TX_REQ_W4AP_CONFIRM, 210
 - TRDP_ST_TX_REQUEST_ARM, 209
 - TRDP_ST_TX_REQUEST_W4REPLY, 210
 - TRDP_TIMED_OUT, 210
- trdp_proto.h
 - TRDP_MSG_MC, 214
 - TRDP_MSG_ME, 214
 - TRDP_MSG_MN, 214
 - TRDP_MSG_MP, 214
 - TRDP_MSG_MQ, 214
 - TRDP_MSG_MR, 214
 - TRDP_MSG_PD, 214
 - TRDP_MSG_PE, 214
 - TRDP_MSG_PP, 214
 - TRDP_MSG_PR, 214
 - TRDP_PULL_SUB
 - trdp_private.h, 210
 - TRDP_QUEUE_ERR
 - trdp_types.h, 232
 - TRDP_QUEUE_FULL_ERR
 - trdp_types.h, 232
 - TRDP_REAL32
 - trdp_types.h, 231
 - TRDP_REAL64
 - trdp_types.h, 231
 - TRDP_RED_FOLLOWER
 - trdp_types.h, 233
 - TRDP_RED_LEADER
 - trdp_types.h, 233
 - TRDP_REDUNDANT
 - trdp_private.h, 210
 - TRDP_REPLYTO_ERR
 - trdp_types.h, 232
 - TRDP_REQ_2B_SENT
 - trdp_private.h, 210
 - TRDP_REQCONFIRMTO_ERR
 - trdp_types.h, 232
 - TRDP_SEMA_ERR
 - trdp_types.h, 232
 - TRDP_SESSION_ABORT_ERR
 - trdp_types.h, 232
 - TRDP SOCK_ERR
 - trdp_types.h, 232
 - TRDP SOCK_MD_TCP
 - trdp_private.h, 210
 - TRDP SOCK_MD_UDP
 - trdp_private.h, 210
 - TRDP SOCK_PD
 - trdp_private.h, 210
 - TRDP_ST_NONE
 - trdp_private.h, 209
 - TRDP_ST_RX_CONF_RECEIVED
 - trdp_private.h, 210
 - TRDP_ST_RX_NOTIFY_RECEIVED
 - trdp_private.h, 210
 - TRDP_ST_RX_READY
 - trdp_private.h, 210
 - TRDP_ST_RX_REPLY_SENT
 - trdp_private.h, 210
 - TRDP_ST_RX_REPLYQUERY_W4C
 - trdp_private.h, 210
 - TRDP_ST_RX_REQ_W4AP_REPLY
 - trdp_private.h, 210
 - TRDP_ST_TX_CONFIRM_ARM

- trdp_private.h, [210](#)
- TRDP_ST_TX_NOTIFY_ARM
 - trdp_private.h, [209](#)
- TRDP_ST_TX_REPLY_ARM
 - trdp_private.h, [209](#)
- TRDP_ST_TX_REPLY_RECEIVED
 - trdp_private.h, [210](#)
- TRDP_ST_TX_REPLYQUERY_ARM
 - trdp_private.h, [209](#)
- TRDP_ST_TX_REQ_W4AP_CONFIRM
 - trdp_private.h, [210](#)
- TRDP_ST_TX_REQUEST_ARM
 - trdp_private.h, [209](#)
- TRDP_ST_TX_REQUEST_W4REPLY
 - trdp_private.h, [210](#)
- TRDP_STATE_ERR
 - trdp_types.h, [232](#)
- TRDP_THREAD_ERR
 - trdp_types.h, [232](#)
- TRDP_TIMED_OUT
 - trdp_private.h, [210](#)
- TRDP_TIMEDATE32
 - trdp_types.h, [231](#)
- TRDP_TIMEDATE48
 - trdp_types.h, [231](#)
- TRDP_TIMEDATE64
 - trdp_types.h, [231](#)
- TRDP_TIMEOUT_ERR
 - trdp_types.h, [232](#)
- TRDP_TO_DEFAULT
 - trdp_types.h, [233](#)
- TRDP_TO_KEEP_LAST_VALUE
 - trdp_types.h, [233](#)
- TRDP_TO_SET_TO_ZERO
 - trdp_types.h, [233](#)
- TRDP_TOPO_ERR
 - trdp_types.h, [232](#)
- TRDP_TYPE_MAX
 - trdp_types.h, [231](#)
- trdp_types.h
 - TRDP_APP_CONFIRMTO_ERR, [232](#)
 - TRDP_APP_REPLYTO_ERR, [232](#)
 - TRDP_APP_TIMEOUT_ERR, [232](#)
 - TRDP_BLOCK_ERR, [232](#)
 - TRDP_BOOLEAN, [231](#)
 - TRDP_CHAR8, [231](#)
 - TRDP_COMID_ERR, [232](#)
 - TRDP_CONFIRMTO_ERR, [232](#)
 - TRDP_CRC_ERR, [232](#)
 - TRDP_FLAGS_CALLBACK, [232](#)
 - TRDP_FLAGS_DEFAULT, [232](#)
 - TRDP_FLAGS_MARSHALL, [232](#)
 - TRDP_FLAGS_NONE, [232](#)
 - TRDP_FLAGS_TCP, [232](#)
 - TRDP_INIT_ERR, [232](#)
 - TRDP_INT16, [231](#)
 - TRDP_INT32, [231](#)
 - TRDP_INT64, [231](#)
 - TRDP_INT8, [231](#)
 - TRDP_INTEGRATION_ERR, [232](#)
 - TRDP_IO_ERR, [232](#)
 - TRDP_MEM_ERR, [232](#)
 - TRDP_MUTEX_ERR, [232](#)
 - TRDP_NO_ERR, [231](#)
 - TRDP_NOCONN_ERR, [232](#)
 - TRDP_NODATA_ERR, [232](#)
 - TRDP_NOINIT_ERR, [232](#)
 - TRDP_NOLIST_ERR, [232](#)
 - TRDP_NOPUB_ERR, [232](#)
 - TRDP_NOSESSION_ERR, [232](#)
 - TRDP_NOSUB_ERR, [232](#)
 - TRDP_OPTION_BLOCK, [233](#)
 - TRDP_OPTION_TRAFFIC_SHAPING, [233](#)
 - TRDP_PACKET_ERR, [232](#)
 - TRDP_PARAM_ERR, [231](#)
 - TRDP_QUEUE_ERR, [232](#)
 - TRDP_QUEUE_FULL_ERR, [232](#)
 - TRDP_REAL32, [231](#)
 - TRDP_REAL64, [231](#)
 - TRDP_RED_FOLLOWER, [233](#)
 - TRDP_RED_LEADER, [233](#)
 - TRDP_REPLYTO_ERR, [232](#)
 - TRDP_REQCONFIRMTO_ERR, [232](#)
 - TRDP_SEMA_ERR, [232](#)
 - TRDP_SESSION_ABORT_ERR, [232](#)
 - TRDP_SOCKET_ERR, [232](#)
 - TRDP_STATE_ERR, [232](#)
 - TRDP_THREAD_ERR, [232](#)
 - TRDP_TIMEDATE32, [231](#)
 - TRDP_TIMEDATE48, [231](#)
 - TRDP_TIMEDATE64, [231](#)
 - TRDP_TIMEOUT_ERR, [232](#)
 - TRDP_TO_DEFAULT, [233](#)
 - TRDP_TO_KEEP_LAST_VALUE, [233](#)
 - TRDP_TO_SET_TO_ZERO, [233](#)
 - TRDP_TOPO_ERR, [232](#)
 - TRDP_TYPE_MAX, [231](#)
 - TRDP_UINT16, [231](#)
 - TRDP_UINT32, [231](#)
 - TRDP_UINT64, [231](#)
 - TRDP_UINT8, [231](#)
 - TRDP_UNKNOWN_ERR, [232](#)
 - TRDP_UTF16, [231](#)
 - TRDP_WIRE_ERR, [232](#)
- TRDP_UINT16
 - trdp_types.h, [231](#)
- TRDP_UINT32
 - trdp_types.h, [231](#)

- TRDP_UINT64
 - trdp_types.h, 231
- TRDP_UINT8
 - trdp_types.h, 231
- TRDP_UNKNOWN_ERR
 - trdp_types.h, 232
- TRDP_UTF16
 - trdp_types.h, 231
- TRDP_WIRE_ERR
 - trdp_types.h, 232
- TRDP_CAR_INFO_T, 20
 - orient, 21
 - pDevInfo, 21
- trdp_closeMDSessions
 - trdp_mdcom.c, 175
 - trdp_mdcom.h, 183
- TRDP_COMID_DSID_MAP_T, 22
- TRDP_COMID_ECHO
 - trdp_proto.h, 213
- TRDP_CST_INFO_T, 23
 - orient, 24
 - owner, 24
 - pCarInfo, 24
 - pFctInfo, 24
- TRDP_DATA_TYPE_T
 - trdp_types.h, 231
- TRDP_DATASET, 25
- TRDP_DATASET_ELEMENT_T, 26
 - type, 26
- TRDP_DBG_CONFIG_T, 27
- TRDP_DBG_OPTION_T
 - tau_xml.h, 107
- TRDP_DEST_URI_SIZE
 - trdp_proto.h, 213
- TRDP_DEVICE_INFO_T, 28
 - orient, 29
- trdp_dllmain.c, 111
- TRDP_ERR_T
 - trdp_types.h, 231
- TRDP_FCT_INFO_T, 30
- TRDP_FCT_T
 - tau_tti.h, 94
- TRDP_FLAGS_T
 - trdp_types.h, 232
- trdp_getSeqCnt
 - trdp_utils.c, 236
 - trdp_utils.h, 246
- trdp_getTCPSocket
 - trdp_mdcom.c, 175
 - trdp_mdcom.h, 184
- TRDP_HANDLE, 31
- trdp_if.c, 112
 - tlc_closeSession, 115
 - tlc_getInterval, 115
 - tlc_getVersion, 116
 - tlc_getVersionString, 116
 - tlc_init, 116
 - tlc_openSession, 117
 - tlc_process, 119
 - tlc_reinitSession, 120
 - tlc_setTopoCount, 121
 - tlc_terminate, 121
 - tlp_get, 122
 - tlp_getRedundant, 123
 - tlp_publish, 124
 - tlp_put, 126
 - tlp_request, 127
 - tlp_setRedundant, 128
 - tlp_subscribe, 129
 - tlp_unpublish, 130
 - tlp_unsubscribe, 131
 - trdp_isValidSession, 131
 - trdp_sessionQueue, 132
- trdp_if.h, 133
 - trdp_isValidSession, 134
 - trdp_sessionQueue, 134
- trdp_if_light.h, 135
 - tlc_closeSession, 139
 - tlc_freeBuf, 140
 - tlc_getInterval, 140
 - tlc_getJoinStatistics, 141
 - tlc_getListStatistics, 142
 - tlc_getPubStatistics, 143
 - tlc_getRedStatistics, 144
 - tlc_getStatistics, 144
 - tlc_getSubsStatistics, 145
 - tlc_getVersion, 146
 - tlc_getVersionString, 146
 - tlc_init, 147
 - tlc_openSession, 147
 - tlc_process, 150
 - tlc_reinitSession, 151
 - tlc_resetStatistics, 152
 - tlc_setTopoCount, 153
 - tlc_terminate, 153
 - tlm_abortSession, 154
 - tlm_addListener, 154
 - tlm_confirm, 155
 - tlm_delListener, 156
 - tlm_notify, 156
 - tlm_reply, 157
 - tlm_replyErr, 157
 - tlm_replyQuery, 158
 - tlm_request, 159
 - tlp_get, 160
 - tlp_getRedundant, 161
 - tlp_publish, 162
 - tlp_put, 164

- tlp_request, 165
- tlp_setRedundant, 167
- tlp_subscribe, 168
- tlp_unpublish, 170
- tlp_unsubscribe, 171
- TRDP_INAUG_STATE_T
 - tau_tti.h, 95
- trdp_initSockets
 - trdp_utils.c, 237
 - trdp_utils.h, 246
- trdp_initStats
 - trdp_stats.c, 220
 - trdp_stats.h, 223
- trdp_initUncompletedTCP
 - trdp_utils.h, 247
- TRDP_IP_ADDR_T
 - trdp_types.h, 229
- trdp_isAddressed
 - trdp_utils.c, 237
 - trdp_utils.h, 247
- trdp_isRcvSeqCnt
 - trdp_utils.c, 237
 - trdp_utils.h, 247
- trdp_isValidSession
 - trdp_if.c, 131
 - trdp_if.h, 134
- TRDP_LIST_STATISTICS_T, 32
- TRDP_MARSHALL_CONFIG_T, 33
- TRDP_MARSHALL_T
 - trdp_types.h, 229
- TRDP_MAX_FILE_NAME_LEN
 - trdp_proto.h, 213
- TRDP_MAX_LABEL_LEN
 - trdp_proto.h, 213
- TRDP_MAX_URI_HOST_LEN
 - trdp_proto.h, 214
- TRDP_MAX_URI_LEN
 - trdp_proto.h, 214
- TRDP_MAX_URI_USER_LEN
 - trdp_proto.h, 214
- TRDP_MD_CALLBACK_T
 - trdp_types.h, 229
- TRDP_MD_CONFIG_T, 34
- TRDP_MD_ELE_ST_T
 - trdp_private.h, 209
- TRDP_MD_INFO_T, 36
 - msgType, 37
- TRDP_MD_STATISTICS_T, 38
- TRDP_MD_TCP, 40
- trdp_mdCheck
 - trdp_mdcom.c, 175
- trdp_mdCheckListenSocks
 - trdp_mdcom.c, 176
 - trdp_mdcom.h, 184
- trdp_mdCheckPending
 - trdp_mdcom.c, 177
 - trdp_mdcom.h, 185
- trdp_mdCheckTimeouts
 - trdp_mdcom.c, 178
 - trdp_mdcom.h, 186
- trdp_mdcom.c, 173
 - trdp_closeMDSessions, 175
 - trdp_getTCPSocket, 175
 - trdp_mdCheck, 175
 - trdp_mdCheckListenSocks, 176
 - trdp_mdCheckPending, 177
 - trdp_mdCheckTimeouts, 178
 - trdp_mdFreeSession, 178
 - trdp_mdRecv, 178
 - trdp_mdRecvPacket, 179
 - trdp_mdSend, 180
 - trdp_mdSendPacket, 180
 - trdp_mdSetSessionTimeout, 181
 - trdp_mdUpdatePacket, 181
- trdp_mdcom.h, 182
 - trdp_closeMDSessions, 183
 - trdp_getTCPSocket, 184
 - trdp_mdCheckListenSocks, 184
 - trdp_mdCheckPending, 185
 - trdp_mdCheckTimeouts, 186
 - trdp_mdFreeSession, 186
 - trdp_mdRecv, 186
 - trdp_mdSend, 187
 - trdp_mdSendPacket, 188
 - trdp_mdSetSessionTimeout, 188
 - trdp_mdUpdatePacket, 189
- trdp_mdFreeSession
 - trdp_mdcom.c, 178
 - trdp_mdcom.h, 186
- trdp_MDQueueAppLast
 - trdp_utils.c, 238
 - trdp_utils.h, 248
- trdp_MDQueueDelElement
 - trdp_utils.c, 238
 - trdp_utils.h, 248
- trdp_MDQueueFindAddr
 - trdp_utils.c, 238
 - trdp_utils.h, 248
- trdp_MDQueueInsFirst
 - trdp_utils.c, 238
 - trdp_utils.h, 248
- trdp_mdRecv
 - trdp_mdcom.c, 178
 - trdp_mdcom.h, 186
- trdp_mdRecvPacket
 - trdp_mdcom.c, 179
- trdp_mdSend
 - trdp_mdcom.c, 180

- trdp_mdcom.h, 187
- trdp_mdSendPacket
 - trdp_mdcom.c, 180
 - trdp_mdcom.h, 188
- trdp_mdSetSessionTimeout
 - trdp_mdcom.c, 181
 - trdp_mdcom.h, 188
- trdp_mdUpdatePacket
 - trdp_mdcom.c, 181
 - trdp_mdcom.h, 189
- TRDP_MEM_CONFIG_T, 41
- TRDP_MEM_STATISTICS_T, 42
- TRDP_MSG_T
 - trdp_proto.h, 214
- TRDP_OPTION_T
 - trdp_types.h, 232
- trdp_packetSizeMD
 - trdp_utils.c, 239
 - trdp_utils.h, 248
- trdp_packetSizePD
 - trdp_utils.c, 239
 - trdp_utils.h, 249
- TRDP_PD_CALLBACK_T
 - trdp_types.h, 230
- TRDP_PD_CONFIG_T, 43
- TRDP_PD_INFO_T, 44
 - msgType, 45
- TRDP_PD_STATISTICS_T, 46
- trdp_pdCheck
 - trdp_pdcom.c, 192
 - trdp_pdcom.h, 200
- trdp_pdCheckListenSocks
 - trdp_pdcom.c, 192
 - trdp_pdcom.h, 200
- trdp_pdCheckPending
 - trdp_pdcom.c, 193
 - trdp_pdcom.h, 201
- trdp_pdcom.c, 190
 - trdp_pdCheck, 192
 - trdp_pdCheckListenSocks, 192
 - trdp_pdCheckPending, 193
 - trdp_pdDataUpdate, 193
 - trdp_pdDistribute, 193
 - trdp_pdHandleTimeOuts, 194
 - trdp_pdInit, 194
 - trdp_pdReceive, 195
 - trdp_pdSend, 196
 - trdp_pdSendQueued, 197
 - trdp_pdUpdate, 197
- trdp_pdcom.h, 198
 - trdp_pdCheck, 200
 - trdp_pdCheckListenSocks, 200
 - trdp_pdCheckPending, 201
 - trdp_pdDataUpdate, 201
 - trdp_pdDistribute, 201
 - trdp_pdHandleTimeOuts, 202
 - trdp_pdInit, 202
 - trdp_pdReceive, 203
 - trdp_pdSend, 204
 - trdp_pdSendQueued, 205
 - trdp_pdUpdate, 205
- trdp_pdDataUpdate
 - trdp_pdcom.c, 193
 - trdp_pdcom.h, 201
- trdp_pdDistribute
 - trdp_pdcom.c, 193
 - trdp_pdcom.h, 201
- trdp_pdHandleTimeOuts
 - trdp_pdcom.c, 194
 - trdp_pdcom.h, 202
- trdp_pdInit
 - trdp_pdcom.c, 194
 - trdp_pdcom.h, 202
- trdp_pdPrepareStats
 - trdp_stats.c, 220
 - trdp_stats.h, 223
- trdp_pdReceive
 - trdp_pdcom.c, 195
 - trdp_pdcom.h, 203
- trdp_pdSend
 - trdp_pdcom.c, 196
 - trdp_pdcom.h, 204
- trdp_pdSendQueued
 - trdp_pdcom.c, 197
 - trdp_pdcom.h, 205
- trdp_pdUpdate
 - trdp_pdcom.c, 197
 - trdp_pdcom.h, 205
- TRDP_PRINT_DBG_T
 - trdp_types.h, 230
- TRDP_PRIV_FLAGS_T
 - trdp_private.h, 210
- trdp_private.h, 206
 - TRDP_MD_ELE_ST_T, 209
 - TRDP_PRIV_FLAGS_T, 210
 - TRDP SOCK_TYPE_T, 210
- TRDP_PROCESS_CONFIG_T, 48
- TRDP_PROP_INFO_T, 49
- trdp_proto.h, 211
 - TRDP_COMID_ECHO, 213
 - TRDP_DEST_URI_SIZE, 213
 - TRDP_MAX_FILE_NAME_LEN, 213
 - TRDP_MAX_LABEL_LEN, 213
 - TRDP_MAX_URI_HOST_LEN, 214
 - TRDP_MAX_URI_LEN, 214
 - TRDP_MAX_URI_USER_LEN, 214
 - TRDP_MSG_T, 214
 - TRDP_STATISTICS_REQUEST_DSID, 214

- TRDP_PUB_STATISTICS_T, 50
 - destAddr, 50
- trdp_queueAppLast
 - trdp_utils.c, 239
 - trdp_utils.h, 249
- trdp_queueDelElement
 - trdp_utils.c, 239
 - trdp_utils.h, 249
- trdp_queueFindComId
 - trdp_utils.c, 239
 - trdp_utils.h, 249
- trdp_queueFindPubAddr
 - trdp_utils.c, 240
 - trdp_utils.h, 249
- trdp_queueFindSubAddr
 - trdp_utils.c, 240
 - trdp_utils.h, 250
- trdp_queueInsFirst
 - trdp_utils.c, 240
 - trdp_utils.h, 250
- TRDP_RED_STATE_T
 - trdp_types.h, 233
- TRDP_RED_STATISTICS_T, 51
- trdp_releaseSocket
 - trdp_utils.c, 240
 - trdp_utils.h, 250
- TRDP_REPLY_STATUS_T
 - trdp_types.h, 233
- trdp_requestSocket
 - trdp_utils.c, 241
 - trdp_utils.h, 251
- TRDP_SDT_DEFAULT_CMTHR
 - tau_xml.c, 102
- TRDP_SDT_PAR_T, 52
- TRDP_SEND_PARAM_T, 53
- TRDP_SESSION, 54
- trdp_sessionQueue
 - trdp_if.c, 132
 - trdp_if.h, 134
- TRDP SOCK_TYPE_T
 - trdp_private.h, 210
- trdp_SockAddJoin
 - trdp_utils.c, 242
- trdp_SockDelJoin
 - trdp_utils.c, 242
- TRDP_SOCKET_TCP, 56
- TRDP_SOCKETS, 57
 - usage, 58
- trdp_SockIsJoined
 - trdp_utils.c, 242
- TRDP_STATISTICS_REQUEST_DSID
 - trdp_proto.h, 214
- TRDP_STATISTICS_T, 59
- trdp_stats.c, 215
- tlc_getJoinStatistics, 216
- tlc_getListStatistics, 217
- tlc_getPubStatistics, 217
- tlc_getRedStatistics, 218
- tlc_getStatistics, 218
- tlc_getSubsStatistics, 219
- tlc_resetStatistics, 220
- trdp_initStats, 220
- trdp_pdPrepareStats, 220
- trdp_UpdateStats, 221
- trdp_stats.h, 222
- trdp_initStats, 223
- trdp_pdPrepareStats, 223
- TRDP_SUBS_STATISTICS_T, 61
 - filterAddr, 61
 - numRecv, 62
 - timeout, 61
 - toBehav, 61
- TRDP_TCP_FD_T, 63
- TRDP_TIME_T
 - trdp_types.h, 230
- TRDP_TO_BEHAVIOR_T
 - trdp_types.h, 233
- TRDP_TRAIN_INFO_T, 64
 - operator, 65
 - pCstInfo, 65
 - topoCnt, 65
- trdp_types.h, 224
- TRDP_DATA_TYPE_T, 231
- TRDP_ERR_T, 231
- TRDP_FLAGS_T, 232
- TRDP_IP_ADDR_T, 229
- TRDP_MARSHALL_T, 229
- TRDP_MD_CALLBACK_T, 229
- TRDP_OPTION_T, 232
- TRDP_PD_CALLBACK_T, 230
- TRDP_PRINT_DBG_T, 230
- TRDP_RED_STATE_T, 233
- TRDP_REPLY_STATUS_T, 233
- TRDP_TIME_T, 230
- TRDP_TO_BEHAVIOR_T, 233
- TRDP_UNMARSHALL_T, 230
- TRDP_UNMARSHALL_T
 - trdp_types.h, 230
- trdp_UpdateStats
 - trdp_stats.c, 221
- trdp_utils.c, 234
 - am_big_endian, 236
 - trdp_getSeqCnt, 236
 - trdp_initSockets, 237
 - trdp_isAddressed, 237
 - trdp_isRcvSeqCnt, 237
 - trdp_MDqueueAppLast, 238
 - trdp_MDqueueDelElement, 238

- trdp_MDqueueFindAddr, 238
- trdp_MDqueueInsFirst, 238
- trdp_packetSizeMD, 239
- trdp_packetSizePD, 239
- trdp_queueAppLast, 239
- trdp_queueDelElement, 239
- trdp_queueFindComId, 239
- trdp_queueFindPubAddr, 240
- trdp_queueFindSubAddr, 240
- trdp_queueInsFirst, 240
- trdp_releaseSocket, 240
- trdp_requestSocket, 241
- trdp_SockAddJoin, 242
- trdp_SockDelJoin, 242
- trdp_SockIsJoined, 242
- trdp_utils.h, 244
 - am_big_endian, 246
 - trdp_getSeqCnt, 246
 - trdp_initSockets, 246
 - trdp_initUncompletedTCP, 247
 - trdp_isAddressed, 247
 - trdp_isRcvSeqCnt, 247
 - trdp_MDqueueAppLast, 248
 - trdp_MDqueueDelElement, 248
 - trdp_MDqueueFindAddr, 248
 - trdp_MDqueueInsFirst, 248
 - trdp_packetSizeMD, 248
 - trdp_packetSizePD, 249
 - trdp_queueAppLast, 249
 - trdp_queueDelElement, 249
 - trdp_queueFindComId, 249
 - trdp_queueFindPubAddr, 249
 - trdp_queueFindSubAddr, 250
 - trdp_queueInsFirst, 250
 - trdp_releaseSocket, 250
 - trdp_requestSocket, 251
- TRDP_VERSION_T, 66
- TRDP_XML_DOC_HANDLE_T, 67
- tv_usec
 - VOS_TIME_T, 69
- type
 - TRDP_DATASET_ELEMENT_T, 26
- usage
 - TRDP_SOCKETS, 58
- VOS_BLOCK_ERR
 - vos_types.h, 373
- VOS_INIT_ERR
 - vos_types.h, 373
- VOS_INTEGRATION_ERR
 - vos_types.h, 373
- VOS_IO_ERR
 - vos_types.h, 373
- VOS_LOG_DBG
 - vos_types.h, 374
- VOS_LOG_ERROR
 - vos_types.h, 374
- VOS_LOG_INFO
 - vos_types.h, 374
- VOS_LOG_WARNING
 - vos_types.h, 374
- VOS_MEM_ERR
 - vos_types.h, 373
- VOS_MUTEX_ERR
 - vos_types.h, 373
- VOS_NO_ERR
 - vos_types.h, 373
- VOS_NOCONN_ERR
 - vos_types.h, 373
- VOS_NODATA_ERR
 - vos_types.h, 373
- VOS_NOINIT_ERR
 - vos_types.h, 373
- VOS_PARAM_ERR
 - vos_types.h, 373
- VOS_QUEUE_ERR
 - vos_types.h, 373
- VOS_QUEUE_FULL_ERR
 - vos_types.h, 373
- VOS_SEMA_ERR
 - vos_types.h, 373
- VOS_SOCKET_ERR
 - vos_types.h, 373
- VOS_THREAD_ERR
 - vos_types.h, 373
- VOS_TIMEOUT_ERR
 - vos_types.h, 373
- vos_types.h
 - VOS_BLOCK_ERR, 373
 - VOS_INIT_ERR, 373
 - VOS_INTEGRATION_ERR, 373
 - VOS_IO_ERR, 373
 - VOS_LOG_DBG, 374
 - VOS_LOG_ERROR, 374
 - VOS_LOG_INFO, 374
 - VOS_LOG_WARNING, 374
 - VOS_MEM_ERR, 373
 - VOS_MUTEX_ERR, 373
 - VOS_NO_ERR, 373
 - VOS_NOCONN_ERR, 373
 - VOS_NODATA_ERR, 373
 - VOS_NOINIT_ERR, 373
 - VOS_PARAM_ERR, 373
 - VOS_QUEUE_ERR, 373
 - VOS_QUEUE_FULL_ERR, 373
 - VOS_SEMA_ERR, 373
 - VOS_SOCKET_ERR, 373

- VOS_THREAD_ERR, [373](#)
- VOS_TIMEOUT_ERR, [373](#)
- VOS_UNKNOWN_ERR, [373](#)
- VOS_UNKNOWN_ERR
 - vos_types.h, [373](#)
- vos_addTime
 - posix/vos_thread.c, [338](#)
 - vos_thread.h, [360](#)
 - windows/vos_thread.c, [349](#)
- vos_bsearch
 - vos_mem.c, [255](#)
 - vos_mem.h, [264](#)
- vos_clearTime
 - posix/vos_thread.c, [338](#)
 - vos_thread.h, [360](#)
 - windows/vos_thread.c, [349](#)
- vos_cmpTime
 - posix/vos_thread.c, [338](#)
 - vos_thread.h, [360](#)
 - windows/vos_thread.c, [349](#)
- vos_crc32
 - vos_utils.c, [376](#)
 - vos_utils.h, [379](#)
- vos_divTime
 - posix/vos_thread.c, [338](#)
 - vos_thread.h, [360](#)
 - windows/vos_thread.c, [349](#)
- vos_dottedIP
 - posix/vos_sock.c, [288](#)
 - vos_sock.h, [316](#)
 - windows/vos_sock.c, [302](#)
- VOS_ERR_T
 - vos_types.h, [373](#)
- vos_getFreeThreadHandle
 - windows/vos_thread.c, [350](#)
- vos_getInterfaces
 - posix/vos_sock.c, [288](#)
 - vos_sock.h, [316](#)
 - windows/vos_sock.c, [302](#)
- vos_getMacAddress
 - posix/vos_sock.c, [288](#)
- vos_getTime
 - posix/vos_thread.c, [338](#)
 - vos_thread.h, [361](#)
 - windows/vos_thread.c, [350](#)
- vos_getTimeStamp
 - posix/vos_thread.c, [339](#)
 - vos_thread.h, [361](#)
 - windows/vos_thread.c, [350](#)
- vos_getUuid
 - posix/vos_thread.c, [339](#)
 - vos_thread.h, [361](#)
 - windows/vos_thread.c, [350](#)
- vos_htonl
 - posix/vos_sock.c, [288](#)
 - vos_sock.h, [317](#)
 - windows/vos_sock.c, [302](#)
- vos_htons
 - posix/vos_sock.c, [289](#)
 - vos_sock.h, [317](#)
 - windows/vos_sock.c, [303](#)
- vos_init
 - vos_types.h, [374](#)
 - vos_utils.c, [376](#)
- vos_initRuntimeConsts
 - vos_utils.c, [376](#)
- vos_ipDotted
 - posix/vos_sock.c, [289](#)
 - vos_sock.h, [318](#)
 - windows/vos_sock.c, [303](#)
- vos_isBigEndian
 - vos_utils.c, [376](#)
- vos_isMulticast
 - posix/vos_sock.c, [289](#)
 - vos_sock.h, [318](#)
 - windows/vos_sock.c, [303](#)
- VOS_LOG_T
 - vos_types.h, [373](#)
- VOS_MAX_ERR_STR_SIZE
 - vos_utils.h, [379](#)
- VOS_MAX_FRMT_SIZE
 - vos_utils.h, [379](#)
- VOS_MAX_PRNT_STR_SIZE
 - vos_utils.h, [379](#)
- VOS_MAX_SOCKET_CNT
 - vos_sock.h, [316](#)
- vos_mem.c, [253](#)
 - vos_bsearch, [255](#)
 - vos_memAlloc, [255](#)
 - vos_memCount, [255](#)
 - vos_memDelete, [256](#)
 - vos_memFree, [256](#)
 - vos_memInit, [257](#)
 - vos_mutexLocalCreate, [257](#)
 - vos_mutexLocalDelete, [258](#)
 - vos_qsort, [258](#)
 - vos_queueCreate, [258](#)
 - vos_queueDestroy, [259](#)
 - vos_queueReceive, [259](#)
 - vos_queueSend, [260](#)
 - vos_strncpy, [260](#)
 - vos_strncmp, [261](#)
- vos_mem.h, [262](#)
 - vos_bsearch, [264](#)
 - VOS_MEM_BLOCKSIZE, [264](#)
 - VOS_MEM_PREALLOCATE, [264](#)
 - vos_memAlloc, [265](#)
 - vos_memCount, [265](#)

- vos_memDelete, 265
- vos_memFree, 266
- vos_memInit, 266
- vos_qsort, 267
- vos_queueCreate, 267
- vos_queueDestroy, 268
- vos_queueReceive, 269
- vos_queueSend, 270
- vos_strncpy, 270
- vos_strncmp, 271
- VOS_MEM_BLOCKSIZEs
 - vos_mem.h, 264
- VOS_MEM_PREALLOCATE
 - vos_mem.h, 264
- vos_memAlloc
 - vos_mem.c, 255
 - vos_mem.h, 265
- vos_memCount
 - vos_mem.c, 255
 - vos_mem.h, 265
- vos_memDelete
 - vos_mem.c, 256
 - vos_mem.h, 265
- vos_memFree
 - vos_mem.c, 256
 - vos_mem.h, 266
- vos_memInit
 - vos_mem.c, 257
 - vos_mem.h, 266
- vos_mulTime
 - posix/vos_thread.c, 339
 - vos_thread.h, 362
 - windows/vos_thread.c, 350
- vos_mutexCreate
 - posix/vos_thread.c, 339
 - vos_thread.h, 362
 - windows/vos_thread.c, 351
- vos_mutexDelete
 - posix/vos_thread.c, 340
 - vos_thread.h, 363
 - windows/vos_thread.c, 351
- vos_mutexLocalCreate
 - posix/vos_private.h, 273
 - posix/vos_thread.c, 340
 - vos_mem.c, 257
 - windows/vos_private.h, 275
 - windows/vos_thread.c, 351
- vos_mutexLocalDelete
 - posix/vos_private.h, 273
 - posix/vos_thread.c, 340
 - vos_mem.c, 258
 - windows/vos_private.h, 275
 - windows/vos_thread.c, 352
- vos_mutexLock
 - posix/vos_thread.c, 341
 - vos_thread.h, 363
 - windows/vos_thread.c, 352
- vos_mutexTryLock
 - posix/vos_thread.c, 341
 - vos_thread.h, 364
 - windows/vos_thread.c, 352
- vos_mutexUnlock
 - posix/vos_thread.c, 341
 - vos_thread.h, 364
 - windows/vos_thread.c, 353
- vos_ntohl
 - posix/vos_sock.c, 289
 - vos_sock.h, 319
 - windows/vos_sock.c, 303
- vos_ntohs
 - posix/vos_sock.c, 290
 - vos_sock.h, 319
 - windows/vos_sock.c, 304
- VOS_PRINT_DBG_T
 - vos_types.h, 373
- vos_private.h, 272, 274
- vos_qsort
 - vos_mem.c, 258
 - vos_mem.h, 267
- vos_queueCreate
 - vos_mem.c, 258
 - vos_mem.h, 267
- vos_queueDestroy
 - vos_mem.c, 259
 - vos_mem.h, 268
- vos_queueReceive
 - vos_mem.c, 259
 - vos_mem.h, 269
- vos_queueSend
 - vos_mem.c, 260
 - vos_mem.h, 270
- vos_select
 - posix/vos_sock.c, 290
 - vos_sock.h, 319
 - windows/vos_sock.c, 304
- vos_semaCreate
 - posix/vos_thread.c, 341
 - vos_thread.h, 365
 - windows/vos_thread.c, 353
- vos_semaDelete
 - posix/vos_thread.c, 342
 - vos_thread.h, 365
 - windows/vos_thread.c, 353
- vos_semaGive
 - posix/vos_thread.c, 342
 - vos_thread.h, 366
 - windows/vos_thread.c, 354
- vos_semaTake

- posix/vos_thread.c, 342
- vos_thread.h, 366
- windows/vos_thread.c, 354
- vos_shared_mem.c, 276, 279
- vos_shared_mem.h, 282
 - vos_sharedClose, 283
 - vos_sharedOpen, 283
- vos_sharedClose
 - posix/vos_shared_mem.c, 277
 - vos_shared_mem.h, 283
 - windows/vos_shared_mem.c, 280
- vos_sharedOpen
 - posix/vos_shared_mem.c, 277
 - vos_shared_mem.h, 283
 - windows/vos_shared_mem.c, 280
- vos_sock.c, 285, 299
- vos_sock.h, 313
 - vos_dottedIP, 316
 - vos_getInterfaces, 316
 - vos_htonl, 317
 - vos_htons, 317
 - vos_ipDotted, 318
 - vos_isMulticast, 318
 - VOS_MAX_SOCKET_CNT, 316
 - vos_ntohl, 319
 - vos_ntohs, 319
 - vos_select, 319
 - vos_sockAccept, 320
 - vos_sockBind, 321
 - vos_sockClose, 322
 - vos_sockConnect, 322
 - vos_sockGetMAC, 323
 - vos_sockInit, 324
 - vos_sockJoinMC, 324
 - vos_sockLeaveMC, 325
 - vos_sockListen, 326
 - vos_sockOpenTCP, 327
 - vos_sockOpenUDP, 327
 - vos_sockReceiveTCP, 328
 - vos_sockReceiveUDP, 330
 - vos_sockSendTCP, 330
 - vos_sockSendUDP, 331
 - vos_sockSetMulticastIf, 333
 - vos_sockSetOptions, 333
- VOS_SOCKET_OPT_T, 68
 - qos, 68
- vos_sockAccept
 - posix/vos_sock.c, 290
 - vos_sock.h, 320
 - windows/vos_sock.c, 304
- vos_sockBind
 - posix/vos_sock.c, 291
 - vos_sock.h, 321
 - windows/vos_sock.c, 305
- vos_sockClose
 - posix/vos_sock.c, 291
 - vos_sock.h, 322
 - windows/vos_sock.c, 305
- vos_sockConnect
 - posix/vos_sock.c, 292
 - vos_sock.h, 322
 - windows/vos_sock.c, 306
- vos_sockGetMAC
 - posix/vos_sock.c, 292
 - vos_sock.h, 323
 - windows/vos_sock.c, 306
- vos_sockInit
 - posix/vos_sock.c, 293
 - vos_sock.h, 324
 - windows/vos_sock.c, 306
- vos_sockJoinMC
 - posix/vos_sock.c, 293
 - vos_sock.h, 324
 - windows/vos_sock.c, 307
- vos_sockLeaveMC
 - posix/vos_sock.c, 293
 - vos_sock.h, 325
 - windows/vos_sock.c, 307
- vos_sockListen
 - posix/vos_sock.c, 294
 - vos_sock.h, 326
 - windows/vos_sock.c, 308
- vos_sockOpenTCP
 - posix/vos_sock.c, 294
 - vos_sock.h, 327
 - windows/vos_sock.c, 308
- vos_sockOpenUDP
 - posix/vos_sock.c, 295
 - vos_sock.h, 327
 - windows/vos_sock.c, 308
- vos_sockReceiveTCP
 - posix/vos_sock.c, 295
 - vos_sock.h, 328
 - windows/vos_sock.c, 309
- vos_sockReceiveUDP
 - posix/vos_sock.c, 296
 - vos_sock.h, 330
 - windows/vos_sock.c, 309
- vos_sockSendTCP
 - posix/vos_sock.c, 296
 - vos_sock.h, 330
 - windows/vos_sock.c, 310
- vos_sockSendUDP
 - posix/vos_sock.c, 297
 - vos_sock.h, 331
 - windows/vos_sock.c, 311
- vos_sockSetMulticastIf
 - posix/vos_sock.c, 297

- vos_sock.h, 333
 - windows/vos_sock.c, 311
- vos_sockSetOptions
 - posix/vos_sock.c, 298
 - vos_sock.h, 333
 - windows/vos_sock.c, 312
- vos_strncpy
 - vos_mem.c, 260
 - vos_mem.h, 270
- vos_strnicmp
 - vos_mem.c, 261
 - vos_mem.h, 271
- vos_subTime
 - posix/vos_thread.c, 343
 - vos_thread.h, 367
 - windows/vos_thread.c, 354
- vos_thread.c, 335, 346
- vos_thread.h, 357
 - vos_addTime, 360
 - vos_clearTime, 360
 - vos_cmpTime, 360
 - vos_divTime, 360
 - vos_getTime, 361
 - vos_getTimeStamp, 361
 - vos_getUuid, 361
 - vos_mulTime, 362
 - vos_mutexCreate, 362
 - vos_mutexDelete, 363
 - vos_mutexLock, 363
 - vos_mutexTryLock, 364
 - vos_mutexUnlock, 364
 - vos_semaCreate, 365
 - vos_semaDelete, 365
 - vos_semaGive, 366
 - vos_semaTake, 366
 - vos_subTime, 367
 - vos_threadCreate, 367
 - vos_threadDelay, 369
 - vos_threadInit, 369
 - vos_threadIsActive, 369
 - vos_threadTerminate, 370
- vos_threadCreate
 - posix/vos_thread.c, 343
 - vos_thread.h, 367
 - windows/vos_thread.c, 354
- vos_threadDelay
 - posix/vos_thread.c, 344
 - vos_thread.h, 369
 - windows/vos_thread.c, 355
- vos_threadInit
 - posix/vos_thread.c, 344
 - vos_thread.h, 369
 - windows/vos_thread.c, 355
- vos_threadIsActive
 - posix/vos_thread.c, 344
 - vos_thread.h, 369
 - windows/vos_thread.c, 356
- vos_threadTerminate
 - posix/vos_thread.c, 344
 - vos_thread.h, 370
 - windows/vos_thread.c, 356
- VOS_TIME_T, 69
 - tv_usec, 69
- vos_types.h, 371
 - VOS_ERR_T, 373
 - vos_init, 374
 - VOS_LOG_T, 373
 - VOS_PRINT_DBG_T, 373
- vos_utils.c, 375
 - vos_crc32, 376
 - vos_init, 376
 - vos_initRuntimeConsts, 376
 - vos_isBigEndian, 376
- vos_utils.h, 378
 - vos_crc32, 379
 - VOS_MAX_ERR_STR_SIZE, 379
 - VOS_MAX_FRMT_SIZE, 379
 - VOS_MAX_PRNT_STR_SIZE, 379
- windows/vos_private.h
 - vos_mutexLocalCreate, 275
 - vos_mutexLocalDelete, 275
- windows/vos_shared_mem.c
 - vos_sharedClose, 280
 - vos_sharedOpen, 280
- windows/vos_sock.c
 - vos_dottedIP, 302
 - vos_getInterfaces, 302
 - vos_htonl, 302
 - vos_htons, 303
 - vos_ipDotted, 303
 - vos_isMulticast, 303
 - vos_ntohl, 303
 - vos_ntohs, 304
 - vos_select, 304
 - vos_sockAccept, 304
 - vos_sockBind, 305
 - vos_sockClose, 305
 - vos_sockConnect, 306
 - vos_sockGetMAC, 306
 - vos_sockInit, 306
 - vos_sockJoinMC, 307
 - vos_sockLeaveMC, 307
 - vos_sockListen, 308
 - vos_sockOpenTCP, 308
 - vos_sockOpenUDP, 308
 - vos_sockReceiveTCP, 309
 - vos_sockReceiveUDP, 309

- vos_sockSendTCP, [310](#)
- vos_sockSendUDP, [311](#)
- vos_sockSetMulticastIf, [311](#)
- vos_sockSetOptions, [312](#)
- windows/vos_thread.c
 - cyclicThread, [348](#)
 - vos_addTime, [349](#)
 - vos_clearTime, [349](#)
 - vos_cmpTime, [349](#)
 - vos_divTime, [349](#)
 - vos_getFreeThreadHandle, [350](#)
 - vos_getTime, [350](#)
 - vos_getTimeStamp, [350](#)
 - vos_getUuid, [350](#)
 - vos_mulTime, [350](#)
 - vos_mutexCreate, [351](#)
 - vos_mutexDelete, [351](#)
 - vos_mutexLocalCreate, [351](#)
 - vos_mutexLocalDelete, [352](#)
 - vos_mutexLock, [352](#)
 - vos_mutexTryLock, [352](#)
 - vos_mutexUnlock, [353](#)
 - vos_semaCreate, [353](#)
 - vos_semaDelete, [353](#)
 - vos_semaGive, [354](#)
 - vos_semaTake, [354](#)
 - vos_subTime, [354](#)
 - vos_threadCreate, [354](#)
 - vos_threadDelay, [355](#)
 - vos_threadInit, [355](#)
 - vos_threadIsActive, [356](#)
 - vos_threadTerminate, [356](#)