

TCNOpen TRDP

Prototype

Generated by Doxygen 1.5.6

Tue Nov 20 17:53:42 2012

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	13
4.2.2	Field Documentation	13
4.2.2.1	pktFlags	13
4.3	PD_ELE Struct Reference	15
4.3.1	Detailed Description	16
4.4	TAU_MARSHALL_INFO_T Struct Reference	17

4.4.1	Detailed Description	17
4.5	TRDP_CAR_INFO_T Struct Reference	18
4.5.1	Detailed Description	19
4.5.2	Field Documentation	19
4.5.2.1	orient	19
4.5.2.2	pDevInfo	19
4.6	TRDP_COMID_DSID_MAP_T Struct Reference	20
4.6.1	Detailed Description	20
4.7	TRDP_CST_INFO_T Struct Reference	21
4.7.1	Detailed Description	22
4.7.2	Field Documentation	22
4.7.2.1	owner	22
4.7.2.2	orient	22
4.7.2.3	pFctInfo	22
4.7.2.4	pCarInfo	22
4.8	TRDP_DATASET_ELEMENT_T Struct Reference	23
4.8.1	Detailed Description	23
4.8.2	Field Documentation	23
4.8.2.1	type	23
4.9	TRDP_DATASET_T Struct Reference	24
4.9.1	Detailed Description	24
4.10	TRDP_DBG_CONFIG_T Struct Reference	25
4.10.1	Detailed Description	25
4.11	TRDP_DEVICE_INFO_T Struct Reference	26
4.11.1	Detailed Description	27
4.11.2	Field Documentation	27
4.11.2.1	orient	27
4.12	TRDP_FCT_INFO_T Struct Reference	28
4.12.1	Detailed Description	28
4.13	TRDP_HANDLE Struct Reference	29
4.13.1	Detailed Description	29
4.14	TRDP_LIST_STATISTICS_T Struct Reference	30
4.14.1	Detailed Description	30
4.15	TRDP_MARSHALL_CONFIG_T Struct Reference	31
4.15.1	Detailed Description	31
4.16	TRDP_MD_CONFIG_T Struct Reference	32

4.16.1 Detailed Description	32
4.17 TRDP_MD_INFO_T Struct Reference	33
4.17.1 Detailed Description	34
4.17.2 Field Documentation	34
4.17.2.1 msgType	34
4.18 TRDP_MD_STATISTICS_T Struct Reference	35
4.18.1 Detailed Description	36
4.19 TRDP_MEM_CONFIG_T Struct Reference	37
4.19.1 Detailed Description	37
4.20 TRDP_MEM_STATISTICS_T Struct Reference	38
4.20.1 Detailed Description	38
4.21 TRDP_PD_CONFIG_T Struct Reference	39
4.21.1 Detailed Description	39
4.22 TRDP_PD_INFO_T Struct Reference	40
4.22.1 Detailed Description	41
4.22.2 Field Documentation	41
4.22.2.1 msgType	41
4.23 TRDP_PD_STATISTICS_T Struct Reference	42
4.23.1 Detailed Description	43
4.24 TRDP_PROCESS_CONFIG_T Struct Reference	44
4.24.1 Detailed Description	44
4.24.2 Field Documentation	44
4.24.2.1 leaderName	44
4.24.2.2 cycleTime	44
4.24.2.3 priority	45
4.24.2.4 options	45
4.25 TRDP_PROP_INFO_T Struct Reference	46
4.25.1 Detailed Description	46
4.26 TRDP_PUB_STATISTICS_T Struct Reference	47
4.26.1 Detailed Description	47
4.26.2 Field Documentation	47
4.26.2.1 destAddr	47
4.27 TRDP_RED_STATISTICS_T Struct Reference	48
4.27.1 Detailed Description	48
4.28 TRDP_SEND_PARAM_T Struct Reference	49
4.28.1 Detailed Description	49

4.29	TRDP_SESSION Struct Reference	50
4.29.1	Detailed Description	51
4.30	TRDP_SOCKETS Struct Reference	52
4.30.1	Detailed Description	52
4.30.2	Field Documentation	52
4.30.2.1	usage	52
4.31	TRDP_STATISTICS_T Struct Reference	53
4.31.1	Detailed Description	54
4.32	TRDP_SUBS_STATISTICS_T Struct Reference	55
4.32.1	Detailed Description	55
4.32.2	Field Documentation	55
4.32.2.1	filterAddr	55
4.32.2.2	timeout	55
4.32.2.3	toBehav	56
4.32.2.4	numRecv	56
4.33	TRDP_TRAIN_INFO_T Struct Reference	57
4.33.1	Detailed Description	58
4.33.2	Field Documentation	58
4.33.2.1	operator	58
4.33.2.2	topoCnt	58
4.33.2.3	pCstInfo	58
4.34	VOS SOCK_OPT_T Struct Reference	59
4.34.1	Detailed Description	59
4.34.2	Field Documentation	59
4.34.2.1	qos	59
4.35	VOS_TIME_T Struct Reference	60
4.35.1	Detailed Description	60
4.35.2	Field Documentation	60
4.35.2.1	tv_usec	60
5	File Documentation	61
5.1	echoPolling.c File Reference	61
5.1.1	Detailed Description	62
5.1.2	Function Documentation	62
5.1.2.1	dbgOut	62
5.1.2.2	main	63
5.2	echoSelect.c File Reference	65

5.2.1	Detailed Description	65
5.2.2	Function Documentation	66
5.2.2.1	dbgOut	66
5.2.2.2	main	67
5.2.2.3	myPDcallBack	69
5.3	mdManager1.c File Reference	70
5.3.1	Detailed Description	70
5.4	mdManager2.c File Reference	72
5.4.1	Detailed Description	72
5.5	sendHello.c File Reference	74
5.5.1	Detailed Description	74
5.5.2	Function Documentation	75
5.5.2.1	main	75
5.6	tau_addr.h File Reference	77
5.6.1	Detailed Description	79
5.6.2	Function Documentation	79
5.6.2.1	tau_addr2CarId	79
5.6.2.2	tau_addr2CarNo	80
5.6.2.3	tau_addr2CstId	80
5.6.2.4	tau_addr2CstNo	80
5.6.2.5	tau_addr2IecCarNo	81
5.6.2.6	tau_addr2IecCstNo	81
5.6.2.7	tau_addr2Uri	81
5.6.2.8	tau_carNo2Ids	82
5.6.2.9	tau_cstNo2CstId	82
5.6.2.10	tau_getOwnAddr	82
5.6.2.11	tau_getOwnIds	82
5.6.2.12	tau_iecCarNo2Ids	83
5.6.2.13	tau_iecCstNo2CstId	83
5.6.2.14	tau_label2CarId	84
5.6.2.15	tau_label2CarNo	84
5.6.2.16	tau_label2CstId	84
5.6.2.17	tau_label2CstNo	85
5.6.2.18	tau_label2IecCarNo	85
5.6.2.19	tau_label2IecCstNo	85
5.6.2.20	tau_uri2Addr	86

5.7	tau_marshall.h File Reference	87
5.7.1	Detailed Description	88
5.7.2	Typedef Documentation	88
5.7.2.1	tau_calcDatasetSize	88
5.7.2.2	tau_marshallIDs	89
5.7.2.3	tau_unmarshallIDs	89
5.7.3	Function Documentation	89
5.7.3.1	tau_initMarshall	89
5.7.3.2	tau_marshall	90
5.7.3.3	tau_unmarshall	90
5.8	tau_tci.h File Reference	92
5.8.1	Detailed Description	94
5.8.2	Enumeration Type Documentation	94
5.8.2.1	TRDP_FCT_T	94
5.8.2.2	TRDP_INAUG_STATE_T	95
5.8.3	Function Documentation	95
5.8.3.1	tau_getCarDevCnt	95
5.8.3.2	tau_getCarInfo	95
5.8.3.3	tau_getCarOrient	96
5.8.3.4	tau_getCstCarCnt	96
5.8.3.5	tau_getCstFctCnt	96
5.8.3.6	tau_getCstFctInfo	97
5.8.3.7	tau_getCstInfo	97
5.8.3.8	tau_getDevInfo	98
5.8.3.9	tau_getEtbState	98
5.8.3.10	tau_getIecCarOrient	98
5.8.3.11	tau_getTrnCarCnt	99
5.8.3.12	tau_getTrnCstCnt	99
5.8.3.13	tau_getTrnInfo	99
5.9	tau_types.h File Reference	100
5.9.1	Detailed Description	100
5.10	tau_xml.h File Reference	101
5.10.1	Detailed Description	102
5.10.2	Enumeration Type Documentation	102
5.10.2.1	TRDP_DBG_OPTION_T	102
5.10.3	Function Documentation	103

5.10.3.1	tau_readXmlConfig	103
5.10.3.2	tau_readXmlDatasetConfig	103
5.11	trdp_if.c File Reference	105
5.11.1	Detailed Description	108
5.11.2	Function Documentation	108
5.11.2.1	tlc_closeSession	108
5.11.2.2	tlc_getInterval	109
5.11.2.3	tlc_getVersion	109
5.11.2.4	tlc_init	110
5.11.2.5	tlc_openSession	110
5.11.2.6	tlc_process	112
5.11.2.7	tlc_reinitSession	114
5.11.2.8	tlc_setTopoCount	114
5.11.2.9	tlc_terminate	114
5.11.2.10	tlm_addListener	115
5.11.2.11	tlm_confirm	116
5.11.2.12	tlm_delListener	117
5.11.2.13	tlm_notify	117
5.11.2.14	tlm_reply	118
5.11.2.15	tlm_replyErr	119
5.11.2.16	tlm_replyQuery	119
5.11.2.17	tlm_request	120
5.11.2.18	tlp_get	121
5.11.2.19	tlp_getRedundant	122
5.11.2.20	tlp_publish	123
5.11.2.21	tlp_put	124
5.11.2.22	tlp_request	125
5.11.2.23	tlp_setRedundant	126
5.11.2.24	tlp_subscribe	127
5.11.2.25	tlp_unpublish	128
5.11.2.26	tlp_unsubscribe	129
5.11.2.27	trdp_getTopoCount	129
5.11.2.28	trdp_isValidSession	130
5.11.2.29	trdp_sessionQueue	130
5.12	trdp_if.h File Reference	131
5.12.1	Detailed Description	131

5.12.2	Function Documentation	132
5.12.2.1	trdp_getTopoCount	132
5.12.2.2	trdp_isValidSession	132
5.12.2.3	trdp_sessionQueue	133
5.13	trdp_if_light.h File Reference	134
5.13.1	Detailed Description	138
5.13.2	Function Documentation	138
5.13.2.1	tlc_closeSession	138
5.13.2.2	tlc_freeBuf	139
5.13.2.3	tlc_getInterval	139
5.13.2.4	tlc_getJoinStatistics	140
5.13.2.5	tlc_getListStatistics	141
5.13.2.6	tlc_getPubStatistics	142
5.13.2.7	tlc_getRedStatistics	143
5.13.2.8	tlc_getStatistics	143
5.13.2.9	tlc_getSubsStatistics	144
5.13.2.10	tlc_getVersion	145
5.13.2.11	tlc_init	145
5.13.2.12	tlc_openSession	146
5.13.2.13	tlc_process	147
5.13.2.14	tlc_reinitSession	150
5.13.2.15	tlc_resetStatistics	150
5.13.2.16	tlc_setTopoCount	151
5.13.2.17	tlc_terminate	152
5.13.2.18	t1m_abortSession	152
5.13.2.19	t1m_addListener	153
5.13.2.20	t1m_confirm	154
5.13.2.21	t1m_delListener	155
5.13.2.22	t1m_notify	156
5.13.2.23	t1m_reply	157
5.13.2.24	t1m_replyErr	159
5.13.2.25	t1m_replyQuery	160
5.13.2.26	t1m_request	161
5.13.2.27	t1p_get	163
5.13.2.28	t1p_getRedundant	164
5.13.2.29	t1p_publish	165

5.13.2.30	tlp_put	167
5.13.2.31	tlp_request	168
5.13.2.32	tlp_setRedundant	170
5.13.2.33	tlp_subscribe	171
5.13.2.34	tlp_unpublish	173
5.13.2.35	tlp_unsubscribe	174
5.14	trdp_marshall.c File Reference	176
5.14.1	Detailed Description	176
5.15	trdp_mdcom.c File Reference	178
5.15.1	Detailed Description	179
5.15.2	Function Documentation	179
5.15.2.1	trdp_mdCheck	179
5.15.2.2	trdp_mdReceive	179
5.15.2.3	trdp_mdRecv	180
5.15.2.4	trdp_mdSend	181
5.15.2.5	trdp_mdUpdate	181
5.15.2.6	trdp_rcvMD	182
5.15.2.7	trdp_sendMD	182
5.16	trdp_mdcom.h File Reference	183
5.16.1	Detailed Description	183
5.16.2	Function Documentation	184
5.16.2.1	trdp_mdReceive	184
5.16.2.2	trdp_mdSend	185
5.16.2.3	trdp_mdUpdate	185
5.17	trdp_pdcom.c File Reference	187
5.17.1	Detailed Description	188
5.17.2	Function Documentation	188
5.17.2.1	trdp_pdCheck	188
5.17.2.2	trdp_pdDataUpdate	189
5.17.2.3	trdp_pdDistribute	189
5.17.2.4	trdp_pdInit	190
5.17.2.5	trdp_pdReceive	190
5.17.2.6	trdp_pdSend	191
5.17.2.7	trdp_pdSendQueued	192
5.17.2.8	trdp_pdUpdate	192
5.18	trdp_pdcom.h File Reference	193

5.18.1	Detailed Description	194
5.18.2	Function Documentation	194
5.18.2.1	trdp_pdCheck	194
5.18.2.2	trdp_pdDataUpdate	195
5.18.2.3	trdp_pdDistribute	195
5.18.2.4	trdp_pdInit	196
5.18.2.5	trdp_pdReceive	196
5.18.2.6	trdp_pdSend	197
5.18.2.7	trdp_pdSendQueued	198
5.18.2.8	trdp_pdUpdate	198
5.19	trdp_private.h File Reference	199
5.19.1	Detailed Description	201
5.19.2	Enumeration Type Documentation	202
5.19.2.1	TRDP_MD_ELE_ST_T	202
5.19.2.2	TRDP_PRIV_FLAGS_T	202
5.19.2.3	TRDP SOCK_TYPE_T	203
5.20	trdp_stats.c File Reference	204
5.20.1	Detailed Description	205
5.20.2	Function Documentation	205
5.20.2.1	tlc_getJoinStatistics	205
5.20.2.2	tlc_getListStatistics	206
5.20.2.3	tlc_getPubStatistics	206
5.20.2.4	tlc_getRedStatistics	207
5.20.2.5	tlc_getStatistics	207
5.20.2.6	tlc_getSubsStatistics	208
5.20.2.7	tlc_resetStatistics	209
5.20.2.8	trdp_initStats	209
5.20.2.9	trdp_pdPrepareStats	209
5.20.2.10	trdp_UpdateStats	210
5.21	trdp_stats.h File Reference	211
5.21.1	Detailed Description	211
5.21.2	Function Documentation	212
5.21.2.1	trdp_initStats	212
5.21.2.2	trdp_pdPrepareStats	212
5.22	trdp_types.h File Reference	213
5.22.1	Detailed Description	218

5.22.2	Define Documentation	218
5.22.2.1	TRDP_COMID_ECHO	218
5.22.2.2	TRDP_MAX_FILE_NAME_LEN	218
5.22.2.3	TRDP_MAX_LABEL_LEN	218
5.22.2.4	TRDP_MAX_URI_HOST_LEN	219
5.22.2.5	TRDP_MAX_URI_LEN	219
5.22.2.6	TRDP_MAX_URI_USER_LEN	219
5.22.2.7	TRDP_STATISTICS_REQUEST_DSID	219
5.22.3	Typedef Documentation	219
5.22.3.1	TRDP_IP_ADDR_T	219
5.22.3.2	TRDP_MARSHALL_T	219
5.22.3.3	TRDP_MD_CALLBACK_T	220
5.22.3.4	TRDP_PD_CALLBACK_T	220
5.22.3.5	TRDP_PRINT_DBG_T	220
5.22.3.6	TRDP_TIME_T	220
5.22.3.7	TRDP_UNMARSHALL_T	220
5.22.4	Enumeration Type Documentation	221
5.22.4.1	TRDP_DATA_TYPE_T	221
5.22.4.2	TRDP_ERR_T	221
5.22.4.3	TRDP_FLAGS_T	222
5.22.4.4	TRDP_MSG_T	222
5.22.4.5	TRDP_OPTION_T	223
5.22.4.6	TRDP_RED_STATE_T	223
5.22.4.7	TRDP_TO_BEHAVIOR_T	223
5.23	trdp_utils.c File Reference	224
5.23.1	Detailed Description	225
5.23.2	Function Documentation	226
5.23.2.1	am_big_endian	226
5.23.2.2	trdp_getSeqCnt	226
5.23.2.3	trdp_initSockets	226
5.23.2.4	trdp_isRcvSeqCnt	226
5.23.2.5	trdp_MDqueueDelElement	227
5.23.2.6	trdp_MDqueueFindAddr	227
5.23.2.7	trdp_MDqueueInsFirst	227
5.23.2.8	trdp_packetSizePD	228
5.23.2.9	trdp_queueAppLast	228

5.23.2.10	trdp_queueDelElement	228
5.23.2.11	trdp_queueFindComId	228
5.23.2.12	trdp_queueFindPubAddr	228
5.23.2.13	trdp_queueFindSubAddr	229
5.23.2.14	trdp_queueInsFirst	229
5.23.2.15	trdp_releaseSocket	229
5.23.2.16	trdp_requestSocket	230
5.24	trdp_utils.h File Reference	231
5.24.1	Detailed Description	232
5.24.2	Function Documentation	233
5.24.2.1	am_big_endian	233
5.24.2.2	trdp_getSeqCnt	233
5.24.2.3	trdp_initSockets	233
5.24.2.4	trdp_isRcvSeqCnt	234
5.24.2.5	trdp_MDqueueDelElement	234
5.24.2.6	trdp_MDqueueFindAddr	234
5.24.2.7	trdp_MDqueueInsFirst	235
5.24.2.8	trdp_packetSizePD	235
5.24.2.9	trdp_queueAppLast	235
5.24.2.10	trdp_queueDelElement	235
5.24.2.11	trdp_queueFindComId	235
5.24.2.12	trdp_queueFindPubAddr	236
5.24.2.13	trdp_queueFindSubAddr	236
5.24.2.14	trdp_queueInsFirst	236
5.24.2.15	trdp_releaseSocket	236
5.24.2.16	trdp_requestSocket	237
5.25	vos_mem.c File Reference	238
5.25.1	Detailed Description	239
5.25.2	Function Documentation	239
5.25.2.1	vos_bsearch	239
5.25.2.2	vos_memAlloc	240
5.25.2.3	vos_memCount	240
5.25.2.4	vos_memDelete	241
5.25.2.5	vos_memFree	241
5.25.2.6	vos_memInit	241
5.25.2.7	vos_qsort	242

5.26	vos_mem.h File Reference	243
5.26.1	Detailed Description	244
5.26.2	Define Documentation	244
5.26.2.1	VOS_MEM_BLOCKSIZEs	244
5.26.2.2	VOS_MEM_PREALLOCATE	245
5.26.3	Function Documentation	245
5.26.3.1	vos_bsearch	245
5.26.3.2	vos_memAlloc	246
5.26.3.3	vos_memCount	246
5.26.3.4	vos_memDelete	246
5.26.3.5	vos_memFree	247
5.26.3.6	vos_memInit	248
5.26.3.7	vos_qsort	249
5.27	vos_private.h File Reference	250
5.27.1	Detailed Description	250
5.27.2	Function Documentation	251
5.27.2.1	vos_mutexLocalCreate	251
5.27.2.2	vos_mutexLocalDelete	251
5.28	vos_private.h File Reference	252
5.28.1	Detailed Description	252
5.28.2	Function Documentation	253
5.28.2.1	vos_mutexLocalCreate	253
5.28.2.2	vos_mutexLocalDelete	253
5.29	vos_sock.c File Reference	254
5.29.1	Detailed Description	256
5.29.2	Function Documentation	256
5.29.2.1	vos_dottedIP	256
5.29.2.2	vos_htonl	257
5.29.2.3	vos_htons	257
5.29.2.4	vos_ipDotted	257
5.29.2.5	vos_isMulticast	257
5.29.2.6	vos_ntohl	258
5.29.2.7	vos_ntohs	258
5.29.2.8	vos_sockAccept	258
5.29.2.9	vos_sockBind	259
5.29.2.10	vos_sockClose	259

5.29.2.11	<code>vos_sockConnect</code>	259
5.29.2.12	<code>vos_sockGetMAC</code>	260
5.29.2.13	<code>vos_sockInit</code>	260
5.29.2.14	<code>vos_sockJoinMC</code>	260
5.29.2.15	<code>vos_sockLeaveMC</code>	261
5.29.2.16	<code>vos_sockListen</code>	261
5.29.2.17	<code>vos_sockOpenTCP</code>	262
5.29.2.18	<code>vos_sockOpenUDP</code>	262
5.29.2.19	<code>vos_sockReceiveTCP</code>	263
5.29.2.20	<code>vos_sockReceiveUDP</code>	263
5.29.2.21	<code>vos_sockSendTCP</code>	264
5.29.2.22	<code>vos_sockSendUDP</code>	264
5.29.2.23	<code>vos_sockSetOptions</code>	265
5.30	<code>vos_sock.c</code> File Reference	266
5.30.1	Detailed Description	268
5.30.2	Function Documentation	268
5.30.2.1	<code>vos_dottedIP</code>	268
5.30.2.2	<code>vos_htonl</code>	269
5.30.2.3	<code>vos_htons</code>	269
5.30.2.4	<code>vos_ipDotted</code>	269
5.30.2.5	<code>vos_isMulticast</code>	269
5.30.2.6	<code>vos_ntohl</code>	270
5.30.2.7	<code>vos_ntohs</code>	270
5.30.2.8	<code>vos_sockAccept</code>	270
5.30.2.9	<code>vos_sockBind</code>	271
5.30.2.10	<code>vos_sockClose</code>	271
5.30.2.11	<code>vos_sockConnect</code>	271
5.30.2.12	<code>vos_sockGetMAC</code>	272
5.30.2.13	<code>vos_sockInit</code>	272
5.30.2.14	<code>vos_sockJoinMC</code>	272
5.30.2.15	<code>vos_sockLeaveMC</code>	273
5.30.2.16	<code>vos_sockListen</code>	273
5.30.2.17	<code>vos_sockOpenTCP</code>	274
5.30.2.18	<code>vos_sockOpenUDP</code>	274
5.30.2.19	<code>vos_sockReceiveTCP</code>	275
5.30.2.20	<code>vos_sockReceiveUDP</code>	275

5.30.2.21	vos_sockSendTCP	276
5.30.2.22	vos_sockSendUDP	276
5.30.2.23	vos_sockSetOptions	277
5.31	vos_sock.h File Reference	278
5.31.1	Detailed Description	280
5.31.2	Function Documentation	280
5.31.2.1	vos_dottedIP	280
5.31.2.2	vos_htonl	281
5.31.2.3	vos_htons	281
5.31.2.4	vos_ipDotted	282
5.31.2.5	vos_isMulticast	282
5.31.2.6	vos_ntohl	283
5.31.2.7	vos_ntohs	283
5.31.2.8	vos_sockAccept	283
5.31.2.9	vos_sockBind	284
5.31.2.10	vos_sockClose	285
5.31.2.11	vos_sockConnect	286
5.31.2.12	vos_sockGetMAC	287
5.31.2.13	vos_sockInit	288
5.31.2.14	vos_sockJoinMC	288
5.31.2.15	vos_sockLeaveMC	289
5.31.2.16	vos_sockListen	290
5.31.2.17	vos_sockOpenTCP	291
5.31.2.18	vos_sockOpenUDP	292
5.31.2.19	vos_sockReceiveTCP	293
5.31.2.20	vos_sockReceiveUDP	294
5.31.2.21	vos_sockSendTCP	296
5.31.2.22	vos_sockSendUDP	297
5.31.2.23	vos_sockSetOptions	298
5.32	vos_thread.c File Reference	300
5.32.1	Detailed Description	302
5.32.2	Function Documentation	302
5.32.2.1	cyclicThread	302
5.32.2.2	vos_addTime	303
5.32.2.3	vos_clearTime	303
5.32.2.4	vos_cmpTime	303

5.32.2.5	<code>vos_divTime</code>	304
5.32.2.6	<code>vos_getTime</code>	304
5.32.2.7	<code>vos_getTimeStamp</code>	304
5.32.2.8	<code>vos_getUuid</code>	304
5.32.2.9	<code>vos_mulTime</code>	305
5.32.2.10	<code>vos_mutexCreate</code>	305
5.32.2.11	<code>vos_mutexDelete</code>	306
5.32.2.12	<code>vos_mutexLocalCreate</code>	306
5.32.2.13	<code>vos_mutexLocalDelete</code>	306
5.32.2.14	<code>vos_mutexLock</code>	307
5.32.2.15	<code>vos_mutexTryLock</code>	307
5.32.2.16	<code>vos_mutexUnlock</code>	307
5.32.2.17	<code>vos_semaCreate</code>	308
5.32.2.18	<code>vos_semaDelete</code>	308
5.32.2.19	<code>vos_semaGive</code>	308
5.32.2.20	<code>vos_semaTake</code>	309
5.32.2.21	<code>vos_subTime</code>	309
5.32.2.22	<code>vos_threadCreate</code>	309
5.32.2.23	<code>vos_threadDelay</code>	310
5.32.2.24	<code>vos_threadInit</code>	310
5.32.2.25	<code>vos_threadIsActive</code>	310
5.32.2.26	<code>vos_threadTerminate</code>	311
5.33	<code>vos_thread.c</code> File Reference	312
5.33.1	Detailed Description	314
5.33.2	Function Documentation	314
5.33.2.1	<code>cyclicThread</code>	314
5.33.2.2	<code>vos_addTime</code>	315
5.33.2.3	<code>vos_clearTime</code>	315
5.33.2.4	<code>vos_cmpTime</code>	315
5.33.2.5	<code>vos_divTime</code>	315
5.33.2.6	<code>vos_getFreeThreadHandle</code>	316
5.33.2.7	<code>vos_getTime</code>	316
5.33.2.8	<code>vos_getTimeStamp</code>	316
5.33.2.9	<code>vos_getUuid</code>	316
5.33.2.10	<code>vos_mulTime</code>	317
5.33.2.11	<code>vos_mutexCreate</code>	317

5.33.2.12	vos_mutexDelete	318
5.33.2.13	vos_mutexLocalCreate	318
5.33.2.14	vos_mutexLocalDelete	318
5.33.2.15	vos_mutexLock	319
5.33.2.16	vos_mutexTryLock	319
5.33.2.17	vos_mutexUnlock	319
5.33.2.18	vos_subTime	320
5.33.2.19	vos_threadCreate	320
5.33.2.20	vos_threadDelay	321
5.33.2.21	vos_threadInit	321
5.33.2.22	vos_threadIsActive	321
5.33.2.23	vos_threadTerminate	321
5.34	vos_thread.h File Reference	322
5.34.1	Detailed Description	324
5.34.2	Function Documentation	324
5.34.2.1	vos_addTime	324
5.34.2.2	vos_clearTime	325
5.34.2.3	vos_cmpTime	325
5.34.2.4	vos_divTime	326
5.34.2.5	vos_getTime	326
5.34.2.6	vos_getTimeStamp	327
5.34.2.7	vos_getUuid	327
5.34.2.8	vos_mulTime	328
5.34.2.9	vos_mutexCreate	328
5.34.2.10	vos_mutexDelete	329
5.34.2.11	vos_mutexLock	330
5.34.2.12	vos_mutexTryLock	330
5.34.2.13	vos_mutexUnlock	331
5.34.2.14	vos_subTime	332
5.34.2.15	vos_threadCreate	332
5.34.2.16	vos_threadDelay	334
5.34.2.17	vos_threadInit	334
5.34.2.18	vos_threadIsActive	334
5.34.2.19	vos_threadTerminate	335
5.35	vos_types.h File Reference	336
5.35.1	Detailed Description	337

5.35.2	Typedef Documentation	337
5.35.2.1	VOS_PRINT_DBG_T	337
5.35.3	Enumeration Type Documentation	338
5.35.3.1	VOS_ERR_T	338
5.35.3.2	VOS_LOG_T	338
5.35.4	Function Documentation	339
5.35.4.1	vos_init	339
5.36	vos_utils.c File Reference	340
5.36.1	Detailed Description	340
5.36.2	Function Documentation	340
5.36.2.1	vos_crc32	340
5.36.2.2	vos_init	341
5.37	vos_utils.h File Reference	342
5.37.1	Detailed Description	343
5.37.2	Define Documentation	343
5.37.2.1	VOS_MAX_ERR_STR_SIZE	343
5.37.2.2	VOS_MAX_FRMT_SIZE	343
5.37.2.3	VOS_MAX_PRNT_STR_SIZE	343
5.37.3	Function Documentation	343
5.37.3.1	vos_crc32	343

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.

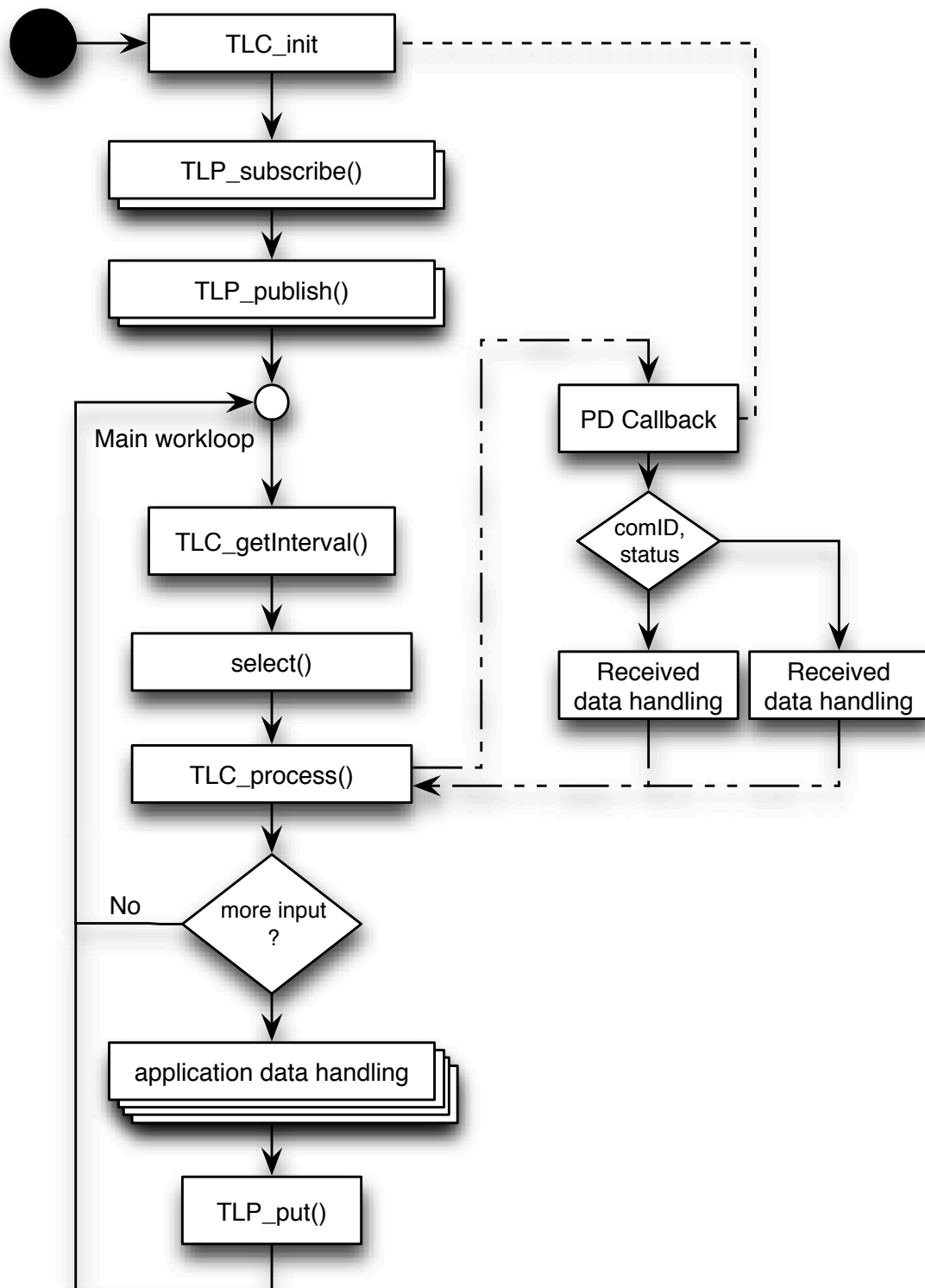


Figure 1.1: Sample client workflow

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 (Queue element for MD packets to send or receive or acknowledge)	12
PD_ELE (Queue element for PD packets to send or receive)	15
TAU_MARSHALL_INFO_T (Marshalling info, used to and from wire)	17
TRDP_CAR_INFO_T (Car information structure)	18
TRDP_COMID_DSID_MAP_T (Dataset element definition)	20
TRDP_CST_INFO_T (Consist information structure)	21
TRDP_DATASET_ELEMENT_T (Dataset element definition)	23
TRDP_DATASET_T (Dataset definition)	24
TRDP_DBG_CONFIG_T (Control for debug output device/file on application level)	25
TRDP_DEVICE_INFO_T (Device information structure)	26
TRDP_FCT_INFO_T (Device information structure)	28
TRDP_HANDLE (Hidden handle definition, used as unique addressing item)	29
TRDP_LIST_STATISTICS_T (Information about a particular MD listener)	30
TRDP_MARSHALL_CONFIG_T (Marshaling/unmarshalling configuration)	31
TRDP_MD_CONFIG_T (Default MD configuration)	32
TRDP_MD_INFO_T (Message data info from received telegram; allows the application to generate responses)	33
TRDP_MD_STATISTICS_T (Structure containing all general MD statistics information)	35
TRDP_MEM_CONFIG_T (Structure describing memory (and its pre-fragmentation))	37
TRDP_MEM_STATISTICS_T (TRDP statistics type definitions)	38
TRDP_PD_CONFIG_T (Default PD configuration)	39
TRDP_PD_INFO_T (Process data info from received telegram; allows the application to generate responses)	40
TRDP_PD_STATISTICS_T (Structure containing all general PD statistics information)	42
TRDP_PROCESS_CONFIG_T (Types to read out the XML configuration)	44
TRDP_PROP_INFO_T (Properties information structure)	46
TRDP_PUB_STATISTICS_T (Table containing particular PD publishing information)	47
TRDP_RED_STATISTICS_T (A table containing PD redundant group information)	48
TRDP_SEND_PARAM_T (Quality/type of service and time to live)	49
TRDP_SESSION (Session/application variables store)	50
TRDP_SOCKETS (Socket item)	52

TRDP_STATISTICS_T (Structure containing all general memory, PD and MD statistics information)	53
TRDP_SUBS_STATISTICS_T (Table containing particular PD subscription information) . . .	55
TRDP_TRAIN_INFO_T (Train information structure)	57
VOS SOCK_OPT_T (Common socket options)	59
VOS_TIME_T (Timer value compatible with timeval / select)	60

Chapter 3

File Index

3.1 File List

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

echoPolling.c (Demo echoing application for TRDP)	61
echoSelect.c (Demo echoing application for TRDP)	65
mdManager1.c (Demo UDPMDCom application for TRDP)	70
mdManager2.c (Demo UDPMDCom application for TRDP)	72
sendHello.c (Demo application for TRDP)	74
tau_addr.h (TRDP utility interface definitions)	77
tau_marshall.h (TRDP utility interface definitions)	87
tau_tci.h (TRDP utility interface definitions)	92
tau_types.h (TRDP utility interface definitions)	100
tau_xml.h (TRDP utility interface definitions)	101
trdp_if.c (Functions for ECN communication)	105
trdp_if.h (Typedefs for TRDP communication)	131
trdp_if_light.h (TRDP Light interface functions (API))	134
trdp_marshall.c (Marshalling functions for TRDP)	176
trdp_mdcom.c (Functions for MD communication)	178
trdp_mdcom.h (Functions for MD communication)	183
trdp_pdcom.c (Functions for PD communication)	187
trdp_pdcom.h (Functions for PD communication)	193
trdp_private.h (Typedefs for TRDP communication)	199
trdp_stats.c (Statistics functions for TRDP communication)	204
trdp_stats.h (Statistics for TRDP communication)	211
trdp_types.h (Typedefs for TRDP communication)	213
trdp_utils.c (Helper functions for TRDP communication)	224
trdp_utils.h (Common utilities for TRDP communication)	231
vos_mem.c (Memory functions)	238
vos_mem.h (Memory and queue functions for OS abstraction)	243
posix/vos_private.h (Private definitions for the OS abstraction layer)	250
windows/vos_private.h (Private definitions for the OS abstraction layer)	252
posix/vos_sock.c (Socket functions)	254
windows/vos_sock.c (Socket functions)	266
vos_sock.h (Typedefs for OS abstraction)	278
posix/vos_thread.c (Multitasking functions)	300
windows/vos_thread.c (Multitasking functions)	312

vos_thread.h (Threading functions for OS abstraction)	322
vos_types.h (Typedefs for OS abstraction)	336
vos_utils.c (Common functions for VOS)	340
vos_utils.h (Typedefs for OS abstraction)	342

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.
- UINT16 [subsAndReserved](#)
first bit (MSB): indicates substitution transmission
- UINT16 [offsetAddress](#)
for process data in traffic store
- UINT32 [replyComId](#)
used in PD request

- UINT32 [replyIpAddress](#)
used for PD request
- 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](#) [MAX_PD_PACKET_SIZE]
data ready to be sent or received (with CRCs)

4.1.1 Detailed Description

TRDP process data header - network order and alignment.

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 file:

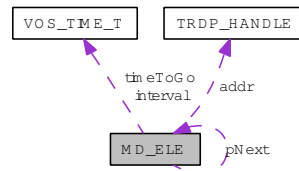
- [trdp_private.h](#)

4.2 MD_ELE Struct Reference

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

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

Collaboration diagram for MD_ELE:



Data Fields

- struct [MD_ELE](#) * [pNext](#)
pointer to next element or NULL
- [TRDP_ADDRESSES_T](#) [addr](#)
handle of publisher/subscriber
- [TRDP_PRIV_FLAGS_T](#) [privFlags](#)
private flags
- [TRDP_FLAGS_T](#) [pktFlags](#)
flags
- [TRDP_TIME_T](#) [interval](#)
time out value for received packets or interval for packets to send (set from ms)
- [TRDP_TIME_T](#) [timeToGo](#)
next time this packet must be sent/rcv
- INT32 [dataSize](#)
net data size
- UINT32 [grossSize](#)
complete packet size (header, data, padding, FCS)
- INT32 [socketIdx](#)
index into the socket list
- [TRDP_MD_ELE_ST_T](#) [stateEle](#)
internal status
- UINT8 [sessionID](#) [16]
UUID as a byte stream.

- MD_HEADER_T [frameHead](#)
Packet header in network byte order.
- UINT8 [data](#) [0]
data ready to be sent (with CRCs)
- UINT32 [comId](#)
filter on incoming MD by comId
- const void * [pUserRef](#)
user reference for call_back from addListener()
- UINT32 [topoCount](#)
set by user: ETB to use, '0' to deactivate
- TRDP_IP_ADDR_T [destIpAddr](#)
filter on incoming MD by destination IP address
- TRDP_URI_USER_T [destURI](#)
filter on incoming MD by destination URI
- struct {
 const void * [pUserRef](#)
 user reference for call_back from addListener()
 UINT32 [comId](#)
 filter on incoming MD by comId
 UINT32 [topoCount](#)
 set by user: ETB to use, '0' to deactivate
 TRDP_IP_ADDR_T [destIpAddr](#)
 filter on incoming MD by destination IP address
 TRDP_FLAGS_T [pktFlags](#)
 marshalling option
 TRDP_URI_USER_T [destURI](#)
 filter on incoming MD by destination URI
 } [listener](#)

 Listener arguments.

4.2.1 Detailed Description

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

4.2.2 Field Documentation

4.2.2.1 TRDP_FLAGS_T MD_ELE::pktFlags

flags

marshalling option

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

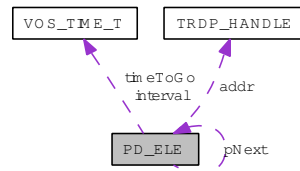
- [trdp_private.h](#)

4.3 PD_ELE Struct Reference

Queue element for PD packets to send or receive.

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

Collaboration diagram for PD_ELE:



Data Fields

- struct [PD_ELE](#) * [pNext](#)
pointer to next element or NULL
- [TRDP_ADDRESSES_T](#) [addr](#)
handle of publisher/subscriber
- [TRDP_IP_ADDR_T](#) [pullIpAddress](#)
In case of pulling a PD this is the requested Ip.
- [UINT32](#) [curSeqCnt](#)
the last sent or received sequence counter
- [UINT32](#) [numRxTx](#)
Counter for received packets (statistics).
- [UINT32](#) [updPkts](#)
Counter for updated 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)
- [INT32 socketIdx](#)
index into the socket list
- `const void *` [userRef](#)
from subscribe()

4.3.1 Detailed Description

Queue element for PD packets to send or receive.

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

- [trdp_private.h](#)

4.4 TAU_MARSHALL_INFO_T Struct Reference

Marshalling info, used to and from wire.

Data Fields

- `UINT8 * pSrc`
source pointer
- `UINT8 * pDst`
destination pointer
- `UINT8 * pDstEnd`
last destination

4.4.1 Detailed Description

Marshalling info, used to and from wire.

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

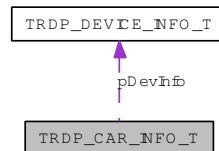
- [trdp_marshall.c](#)

4.5 TRDP_CAR_INFO_T Struct Reference

car information structure.

```
#include <tau_tci.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.

4.5.1 Detailed Description

car information structure.

4.5.2 Field Documentation

4.5.2.1 UINT8 TRDP_CAR_INFO_T::orient

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

to consist

4.5.2.2 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_tci.h](#)

4.6 TRDP_COMID_DSID_MAP_T Struct Reference

Dataset element definition.

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

Data Fields

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

4.6.1 Detailed Description

Dataset element definition.

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

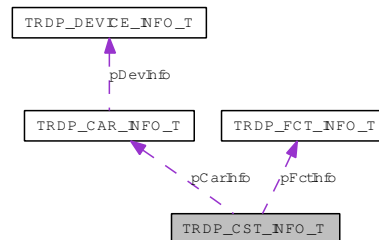
- [trdp_types.h](#)

4.7 TRDP_CST_INFO_T Struct Reference

consist information structure.

```
#include <tau_tci.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.

4.7.1 Detailed Description

consist information structure.

4.7.2 Field Documentation

4.7.2.1 `TRDP_LABEL_T TRDP_CST_INFO_T::owner`

consist owner, e.g.

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

4.7.2.2 `UINT8 TRDP_CST_INFO_T::orient`

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

to train

4.7.2.3 `TRDP_FCT_INFO_T* TRDP_CST_INFO_T::pFctInfo`

Pointer to function info list for application use and convenience.

4.7.2.4 `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_tci.h](#)

4.8 TRDP_DATASET_ELEMENT_T Struct Reference

Dataset element definition.

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

Data Fields

- [UINT32 type](#)
Data type (TRDP_DATA_TYPE_T 1.
- [UINT32 size](#)
Number of items or TDRP_VAR_SIZE (0).

4.8.1 Detailed Description

Dataset element definition.

4.8.2 Field Documentation

4.8.2.1 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:

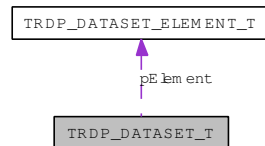
- [trdp_types.h](#)

4.9 TRDP_DATASET_T Struct Reference

Dataset definition.

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

Collaboration diagram for TRDP_DATASET_T:



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_DBG_CONFIG_T Struct Reference

Control for debug output device/file on application level.

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

Data Fields

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

4.10.1 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](#)

4.11 TRDP_DEVICE_INFO_T Struct Reference

device information structure

```
#include <tau_tci.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.

4.11.1 Detailed Description

device information structure

4.11.2 Field Documentation

4.11.2.1 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_tci.h](#)

4.12 TRDP_FCT_INFO_T Struct Reference

device information structure

```
#include <tau_tci.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.

4.12.1 Detailed Description

device information structure

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

- [tau_tci.h](#)

4.13 TRDP_HANDLE Struct Reference

Hidden handle definition, used as unique addressing item.

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

Data Fields

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

4.13.1 Detailed Description

Hidden handle definition, used as unique addressing item.

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

- [trdp_private.h](#)

4.14 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.

4.14.1 Detailed Description

Information about a particular MD listener.

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

- [trdp_types.h](#)

4.15 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.

4.15.1 Detailed Description

Marshaling/unmarshalling configuration.

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

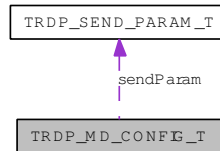
- [trdp_types.h](#)

4.16 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.
- UINT16 [udpPort](#)
Port to be used for UDP MD communication.
- UINT16 [tcpPort](#)
Port to be used for TCP MD communication.

4.16.1 Detailed Description

Default MD configuration.

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

- [trdp_types.h](#)

4.17 TRDP_MD_INFO_T Struct Reference

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

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

Data Fields

- [TRDP_IP_ADDR_T srcIpAddr](#)
source IP address for filtering
- [TRDP_IP_ADDR_T destIpAddr](#)
destination IP address for filtering
- [UINT32 seqCount](#)
sequence counter
- [UINT16 protVersion](#)
Protocol version.
- [TRDP_MSG_T msgType](#)
Protocol ('PD', 'MD', .
- [UINT32 comId](#)
ComID.
- [UINT32 topoCount](#)
received topocount
- [UINT16 numRetries](#)
actual number of retries
- [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 numReplies](#)
actual number of replies for the request
- `const void *` [pUserRef](#)
User reference given with the local call.
- [TRDP_ERR_T resultCode](#)
error code

4.17.1 Detailed Description

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

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

4.17.2 Field Documentation

4.17.2.1 `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](#)

4.18 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

4.18.1 Detailed Description

Structure containing all general MD statistics information.

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

- [trdp_types.h](#)

4.19 TRDP_MEM_CONFIG_T Struct Reference

Structure describing memory (and its pre-fragmentation).

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

Data Fields

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

4.19.1 Detailed Description

Structure describing memory (and its pre-fragmentation).

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

- [trdp_types.h](#)

4.20 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 [preAllocBlockSize](#) [TRDP_MEM_BLK_524288+1]
preallocated memory blocks
- UINT32 [usedBlockSize](#) [TRDP_MEM_BLK_524288+1]
used memory blocks

4.20.1 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:

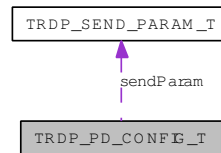
- [trdp_types.h](#)

4.21 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.

4.21.1 Detailed Description

Default PD configuration.

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

- [trdp_types.h](#)

4.22 TRDP_PD_INFO_T Struct Reference

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

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

Data Fields

- [TRDP_IP_ADDR_T srcIpAddr](#)
source IP address for filtering
- [TRDP_IP_ADDR_T destIpAddr](#)
destination IP address for filtering
- [UINT32 seqCount](#)
sequence counter
- [UINT16 protVersion](#)
Protocol version.
- [TRDP_MSG_T msgType](#)
Protocol ('PD', 'MD', .
- [UINT32 comId](#)
ComID.
- [UINT32 topoCount](#)
received topocount
- [BOOL subs](#)
substitution
- [UINT16 offsetAddr](#)
offset address for ladder architecture
- [UINT32 replyComId](#)
ComID for reply (request only).
- [TRDP_IP_ADDR_T replyIpAddr](#)
IP address for reply (request only).
- [const void * pUserRef](#)
User reference given with the local subscribe.
- [TRDP_ERR_T resultCode](#)
error code

4.22.1 Detailed Description

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

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

4.22.2 Field Documentation

4.22.2.1 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](#)

4.23 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

4.23.1 Detailed Description

Structure containing all general PD statistics information.

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

- [trdp_types.h](#)

4.24 TRDP_PROCESS_CONFIG_T Struct Reference

Types to read out the XML configuration.

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

Data Fields

- TRDP_LABEL_T [hostName](#)
Host name.
- TRDP_LABEL_T [leaderName](#)
Leader name dependant on redundanca concept.
- TRDP_IP_ADDR [hostIp](#)
Host IP address.
- TRDP_IP_ADDR [leaderIp](#)
Leader IP address dependant on redundancy concept.
- UINT32 [cycleTime](#)
TRDP main process cycle time in usec.
- UINT32 [priority](#)
TRDP main process priority.
- [TRDP_OPTION_T options](#)
TRDP default options.

4.24.1 Detailed Description

Types to read out the XML configuration.

Various flags/general TRDP options for library initialization.

Configuration of TRDP main process.

4.24.2 Field Documentation

4.24.2.1 TRDP_LABEL_T TRDP_PROCESS_CONFIG_T::leaderName

Leader name dependant on redundanca concept.

Leader name dependant on redundancy concept.

4.24.2.2 UINT32 TRDP_PROCESS_CONFIG_T::cycleTime

TRDP main process cycle time in usec.

TRDP main process cycle time in us.

4.24.2.3 UINT32 TRDP_PROCESS_CONFIG_T::priority

TRDP main process priority.

TRDP main process cycle time (0-255, 0=default, 255=highest).

4.24.2.4 TRDP_OPTION_T TRDP_PROCESS_CONFIG_T::options

TRDP default options.

TRDP options.

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

- [tau_xml.h](#)
- [trdp_types.h](#)

4.25 TRDP_PROP_INFO_T Struct Reference

properties information structure

```
#include <tau_tci.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

4.25.1 Detailed Description

properties information structure

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

- [tau_tci.h](#)

4.26 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.

4.26.1 Detailed Description

Table containing particular PD publishing information.

4.26.2 Field Documentation

4.26.2.1 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](#)

4.27 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.

4.27.1 Detailed Description

A table containing PD redundant group information.

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

- [trdp_types.h](#)

4.28 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).
- `UINT8 retries`
Maximum number of retries for UDP MD if one reply is expected, default should be 2.

4.28.1 Detailed Description

Quality/type of service and time to live.

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

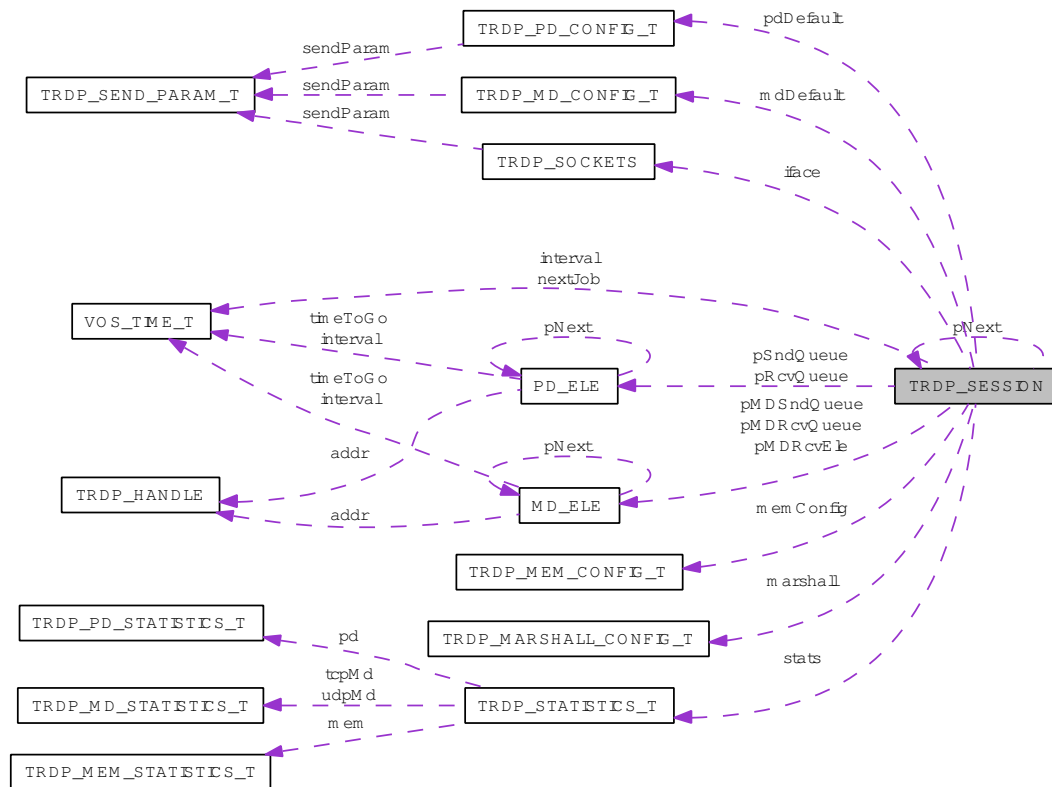
- `trdp_types.h`

4.29 TRDP_SESSION Struct Reference

Session/application variables store.

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

Collaboration diagram for TRDP_SESSION:



Data Fields

- struct `TRDP_SESSION` * `pNext`
Pointer to next session.
- `VOS_MUTEX_T` `mutex`
protect this session
- `TRDP_IP_ADDR_T` `realIP`
Real IP address.
- `TRDP_IP_ADDR_T` `virtualIP`
Virtual IP address.
- `BOOL` `beQuiet`
if set, only react on ownIP requests

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

4.29.1 Detailed Description

Session/application variables store.

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

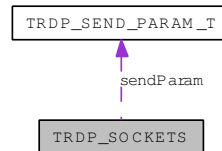
- [trdp_private.h](#)

4.30 TRDP_SOCKETS Struct Reference

Socket item.

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

Collaboration diagram for TRDP_SOCKETS:



Data Fields

- INT32 [sock](#)
vos socket descriptor to use
- [TRDP_IP_ADDR_T](#) [bindAddr](#)
Defines the interface to use.
- [TRDP_SEND_PARAM_T](#) [sendParam](#)
Send parameters.
- [TRDP SOCK_TYPE_T](#) [type](#)
Usage of this socket.
- BOOL [rcvOnly](#)
Used for receiving.
- UINT16 [usage](#)
No.

4.30.1 Detailed Description

Socket item.

4.30.2 Field Documentation

4.30.2.1 UINT16 TRDP_SOCKETS::usage

No.

of current users of this socket

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

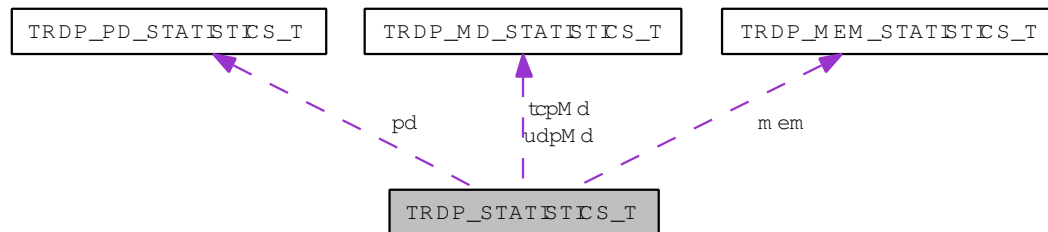
- [trdp_private.h](#)

4.31 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.

4.31.1 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](#)

4.32 TRDP_SUBS_STATISTICS_T Struct Reference

Table containing particular PD subscription information.

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

Data Fields

- [UINT32 comId](#)
Subscribed ComId.
- [TRDP_IP_ADDR_T joinedAddr](#)
Joined IP address.
- [TRDP_IP_ADDR_T filterAddr](#)
Filter IP address, i.e IP address of the sender for this subscription, 0.0.0.0 in case all senders.
- [UINT32 callBack](#)
Reference for call back function if used.
- [UINT32 timeout](#)
Time-out value in us.
- [TRDP_ERR_T status](#)
Receive status information TRDP_NO_ERR, TRDP_TIMEOUT_ERR.
- [TRDP_TO_BEHAVIOR_T toBehav](#)
Behaviour at time-out.
- [UINT32 numRecv](#)
Number of packets received for this subscription.

4.32.1 Detailed Description

Table containing particular PD subscription information.

4.32.2 Field Documentation

4.32.2.1 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.

4.32.2.2 UINT32 TRDP_SUBS_STATISTICS_T::timeout

Time-out value in us.

0 = No time-out supervision

4.32.2.3 TRDP_TO_BEHAVIOR_T TRDP_SUBS_STATISTICS_T::toBehav

Behaviour at time-out.

Set data to zero / keep last value

4.32.2.4 UINT32 TRDP_SUBS_STATISTICS_T::numRecv

Number of packets received for this subscription.

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

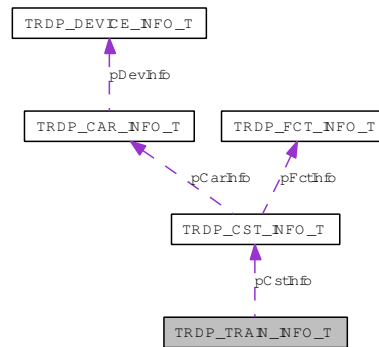
- [trdp_types.h](#)

4.33 TRDP_TRAIN_INFO_T Struct Reference

train information structure.

```
#include <tau_tci.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.

4.33.1 Detailed Description

train information structure.

4.33.2 Field Documentation

4.33.2.1 TRDP_LABEL_T TRDP_TRAIN_INFO_T::operator

Train operator e.g.

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

4.33.2.2 UINT32 TRDP_TRAIN_INFO_T::topoCnt

IEC (i.e.

TCN) topography counter

4.33.2.3 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_tci.h](#)

4.34 VOS SOCK_OPT_T Struct Reference

Common socket options.

```
#include <vos_sock.h>
```

Data Fields

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

4.34.1 Detailed Description

Common socket options.

4.34.2 Field Documentation

4.34.2.1 `UINT8 VOS SOCK_OPT_T::qos`

quality/type of service 0.

..7

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

- `vos_sock.h`

4.35 VOS_TIME_T Struct Reference

Timer value compatible with timeval / select.

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

Data Fields

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

4.35.1 Detailed Description

Timer value compatible with timeval / select.

Relative or absolute date, depending on usage

4.35.2 Field Documentation

4.35.2.1 UINT32 VOS_TIME_T::tv_usec

Micro seconds (max.

value 999999)

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

- [vos_types.h](#)

Chapter 5

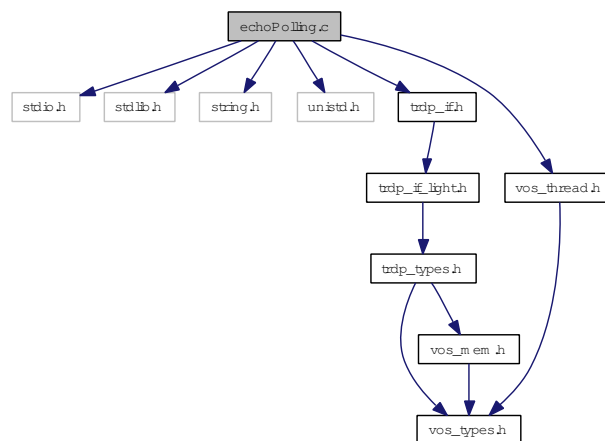
File Documentation

5.1 echoPolling.c File Reference

Demo echoing application for TRDP.

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

Include dependency graph for echoPolling.c:



Functions

- void `dbgOut` (void *pRefCon, `TRDP_LOG_T` category, const CHAR8 *pTime, const CHAR8 *pFile, UINT16 LineNumber, const CHAR8 *pMsgStr)
callback routine for TRDP logging/error output

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

main entry

5.1.1 Detailed Description

Demo echoing application for TRDP.

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

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

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

Id

[echoPolling.c](#) 104 2012-11-02 14:11:53Z 97025

5.1.2 Function Documentation

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

callback routine for TRDP logging/error output

Parameters:

- ← *pRefCon* user supplied context pointer
- ← *category* Log category (Error, Warning, Info etc.)
- ← *pTime* pointer to NULL-terminated string of time stamp
- ← *pFile* pointer to NULL-terminated string of source module
- ← *LineNumber* line
- ← *pMsgStr* pointer to NULL-terminated string

Return values:

none

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

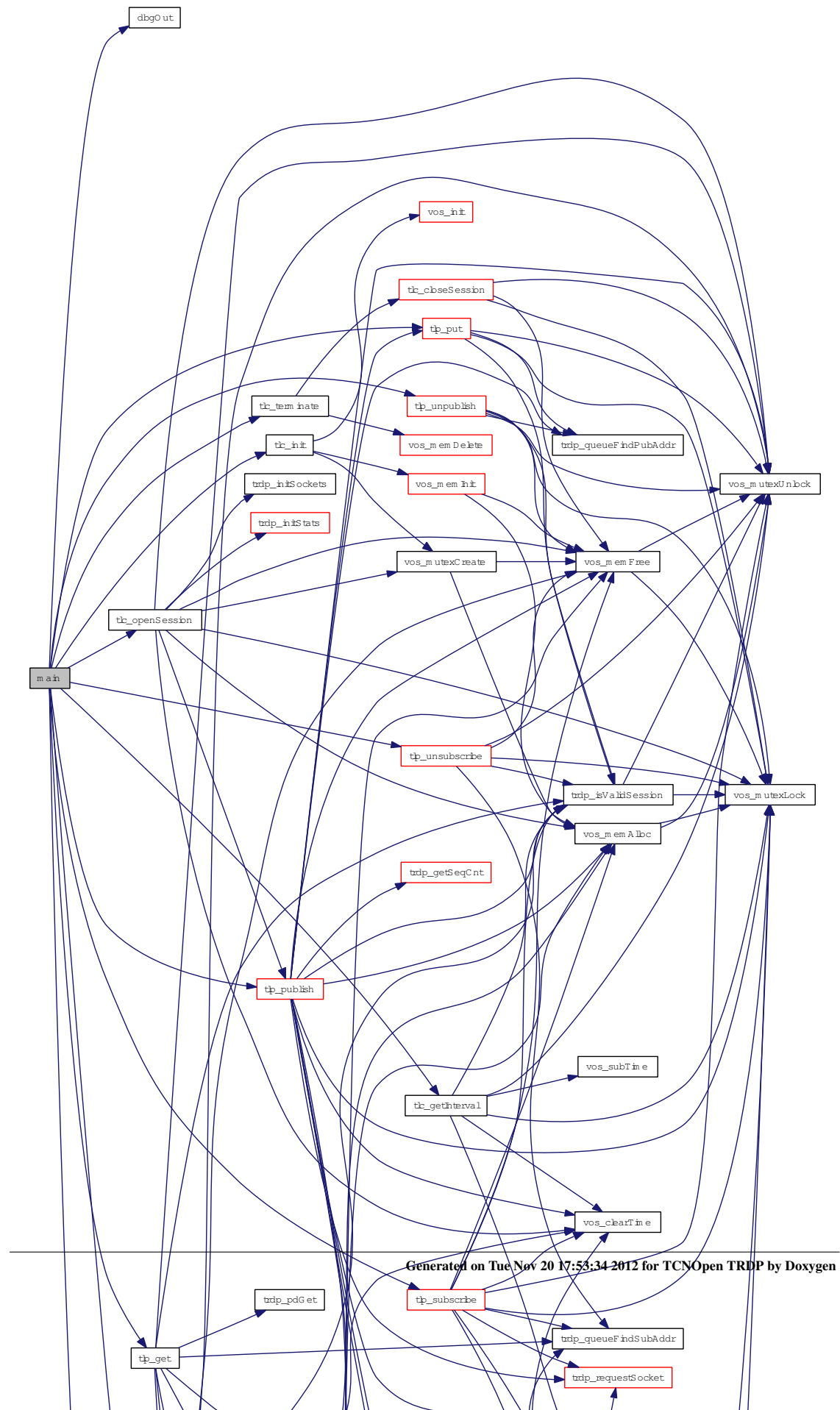
main entry

Return values:

0 no error

1 some error

Here is the call graph for this function:

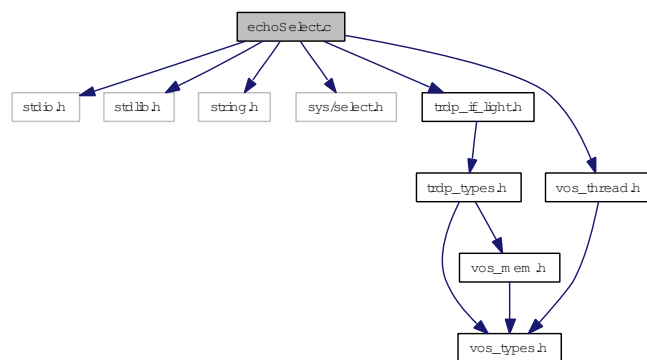


5.2 echoSelect.c File Reference

Demo echoing application for TRDP.

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

Include dependency graph for echoSelect.c:



Functions

- void [dbgOut](#) (void *pRefCon, [TRDP_LOG_T](#) category, const CHAR8 *pTime, const CHAR8 *pFile, UINT16 LineNumber, const CHAR8 *pMsgStr)
callback routine for TRDP logging/error output
- void [myPDcallback](#) (void *pRefCon, const [TRDP_PD_INFO_T](#) *pMsg, UINT8 *pData, UINT32 dataSize)
callback routine for receiving TRDP traffic
- int [main](#) (int argc, char **argv)
main entry

5.2.1 Detailed Description

Demo echoing application for TRDP.

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

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr, NewTec GmbH

Remarks:

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

Id

[echoSelect.c](#) 105 2012-11-04 17:20:25Z 97025

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

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

[echoSelect.c](#) 70 2012-10-19 16:40:23Z 97025

5.2.2 Function Documentation

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

callback routine for TRDP logging/error output

Parameters:

- ← *pRefCon* user supplied context pointer
- ← *category* Log category (Error, Warning, Info etc.)
- ← *pTime* pointer to NULL-terminated string of time stamp
- ← *pFile* pointer to NULL-terminated string of source module
- ← *LineNumber* line
- ← *pMsgStr* pointer to NULL-terminated string

Return values:

none

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

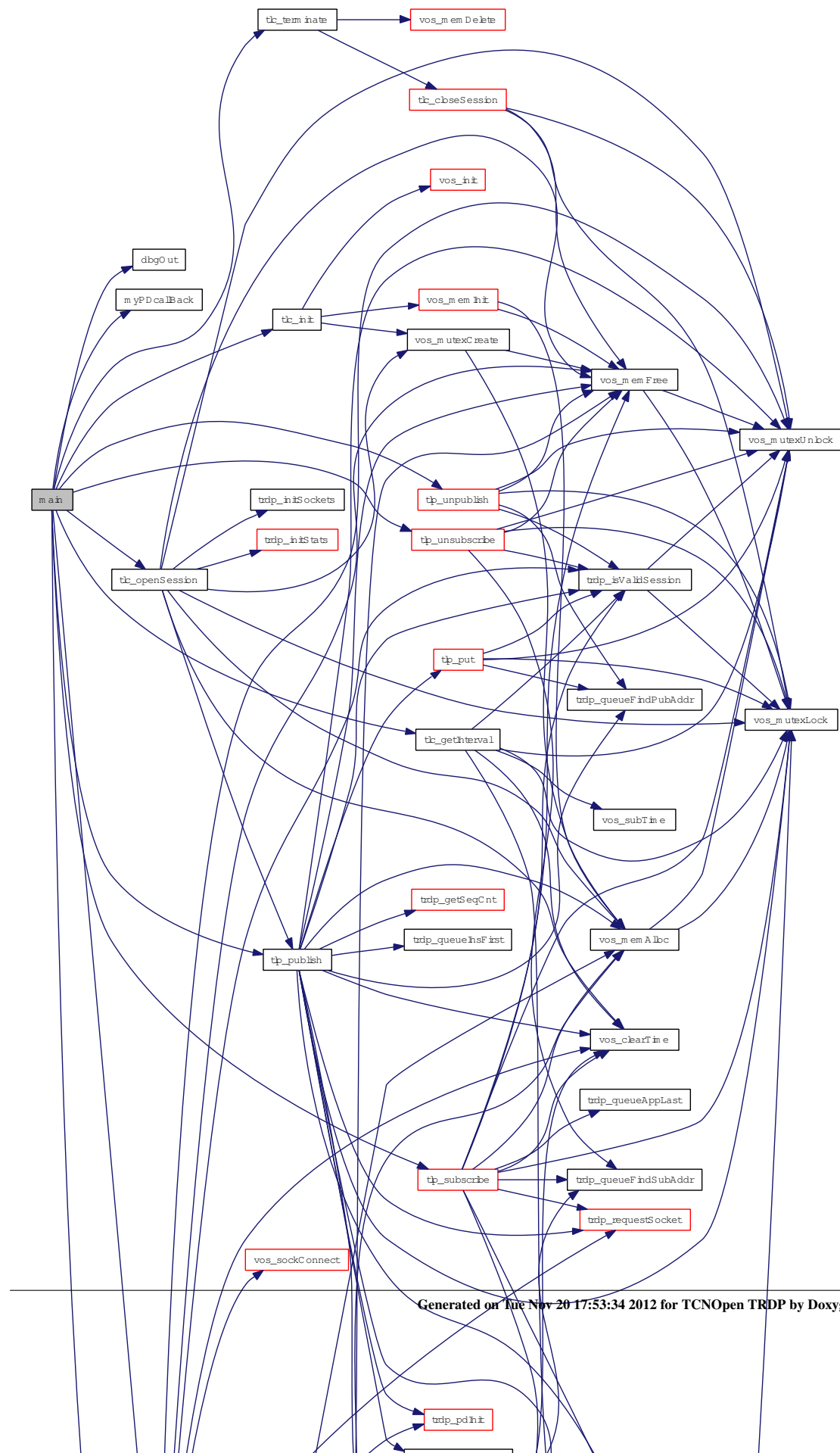
main entry

Return values:

0 no error

1 some error

Here is the call graph for this function:



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

callback routine for receiving TRDP traffic

Parameters:

- ← *pRefCon* user supplied context pointer
- ← *pMsg* pointer to header/packet infos
- ← *pData* pointer to data block
- ← *dataSize* pointer to data size

Return values:

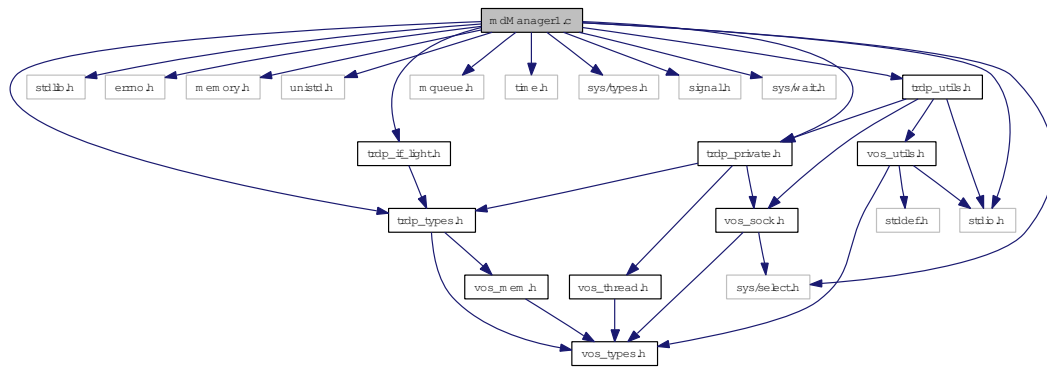
none

5.3 mdManager1.c File Reference

Demo UDPMDCom application for TRDP.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <memory.h>
#include <unistd.h>
#include <sys/select.h>
#include <mqueue.h>
#include <time.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/wait.h>
#include "trdp_types.h"
#include "trdp_if_light.h"
#include "trdp_private.h"
#include "trdp_utils.h"
```

Include dependency graph for mdManager1.c:



5.3.1 Detailed Description

Demo UDPMDCom application for TRDP.

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

Note:

Project: TCNOpen TRDP prototype stack

Author:

Quagred Diego (FAR Systems), Simone Pachera (FAR Systems)

Remarks:

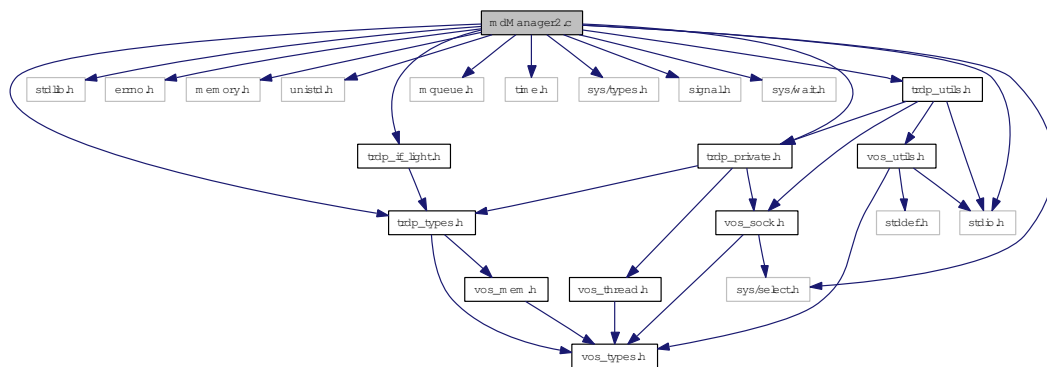
All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, FAR Systems spa, Italy, 2013.

5.4 mdManager2.c File Reference

Demo UDPMDCom application for TRDP.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <memory.h>
#include <unistd.h>
#include <sys/select.h>
#include <mqueue.h>
#include <time.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/wait.h>
#include "trdp_types.h"
#include "trdp_if_light.h"
#include "trdp_private.h"
#include "trdp_utils.h"
```

Include dependency graph for mdManager2.c:



5.4.1 Detailed Description

Demo UDPMDCom application for TRDP.

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

Note:

Project: TCNOpen TRDP prototype stack Version 0.0: d.quagreda (FAR). Initial version. Version 0.1: s.pachera (FAR). Add log to file (12f) to help debug and integration test. Version 0.2: s.pachera (FAR). Add command line interface (cli), add main loop period handling, add test mode

Author:

Quagred Diego (FAR Systems), Simone Pachera (FAR Systems)

Remarks:

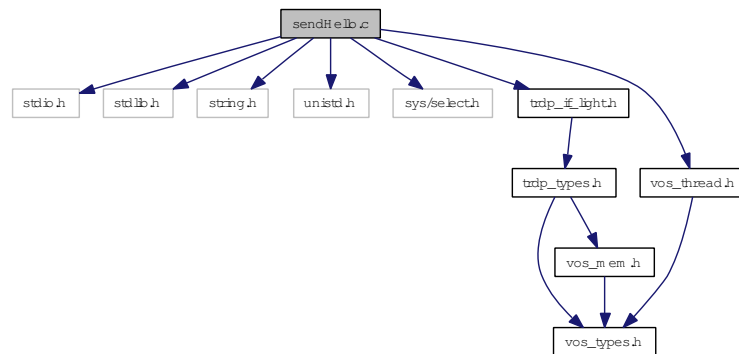
All rights reserved. Reproduction, modification, use or disclosure to third parties without express authority is forbidden, FAR Systems spa, Italy, 2013.

5.5 sendHello.c File Reference

Demo application for TRDP.

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

Include dependency graph for sendHello.c:



Functions

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

5.5.1 Detailed Description

Demo application for TRDP.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Bernd Loehr and Florian Weispfenning, 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

[sendHello.c](#) 104 2012-11-02 14:11:53Z 97025

5.5.2 Function Documentation

5.5.2.1 `int main (int argc, char * argv[])`

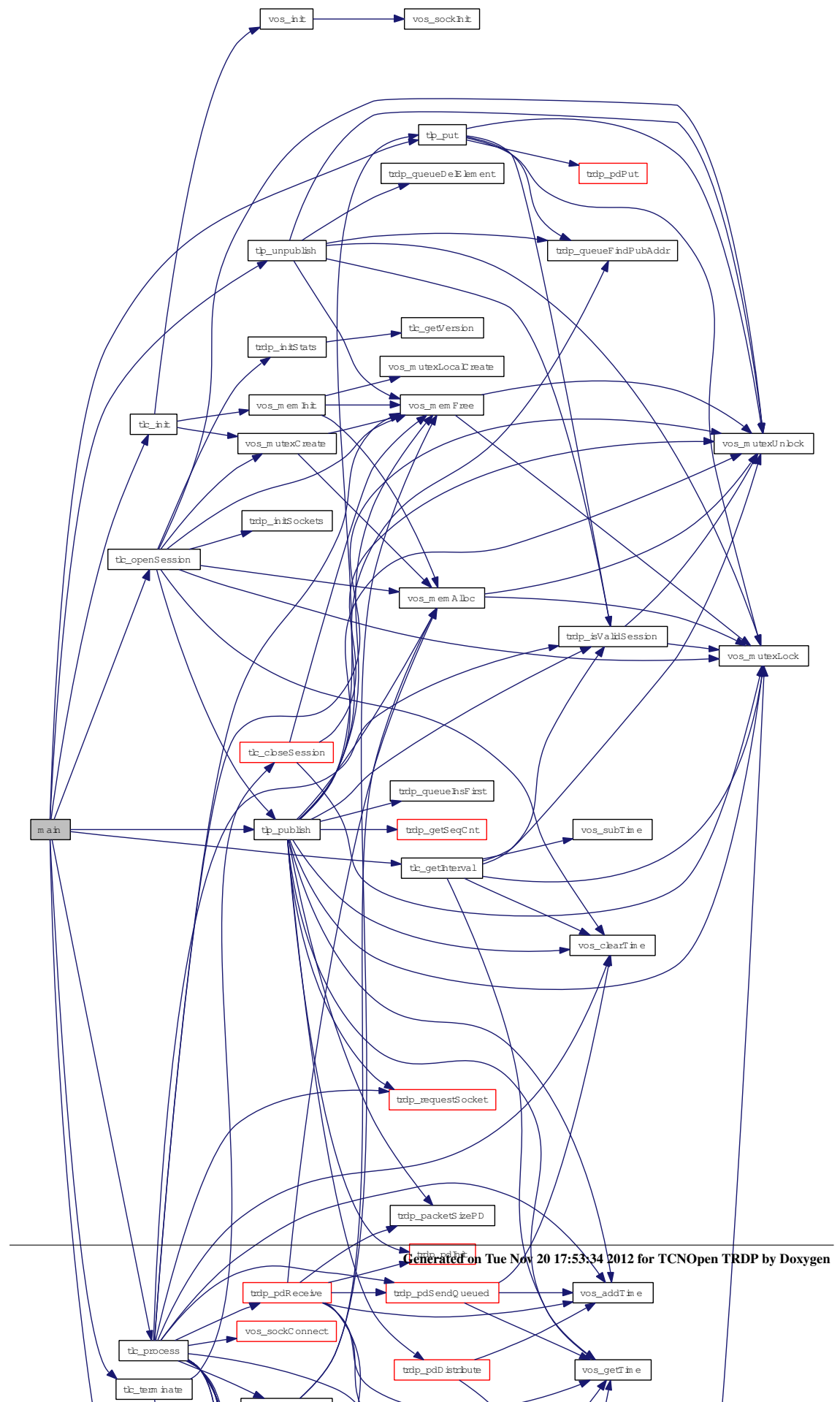
main entry

Return values:

0 no error

1 some error

Here is the call graph for this function:



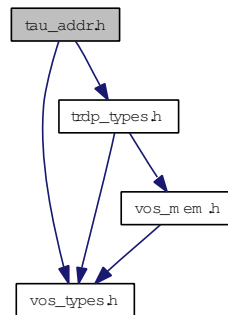
5.6 tau_addr.h File Reference

TRDP utility interface definitions.

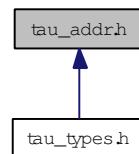
```
#include "vos_types.h"
```

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

Include dependency graph for tau_addr.h:



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



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 URI.
- 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.

5.6.1 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](#) 8 2012-06-06 16:28:19Z 97025

5.6.2 Function Documentation

5.6.2.1 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

5.6.2.2 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

5.6.2.3 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

5.6.2.4 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

5.6.2.5 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

5.6.2.6 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

5.6.2.7 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

5.6.2.8 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

5.6.2.9 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

5.6.2.10 EXT_DECL TRDP_IP_ADDR tau_getOwnAddr (void)

Function to get the own IP address.

Return values:

- own* IP address

5.6.2.11 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

5.6.2.12 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

5.6.2.13 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

5.6.2.14 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

5.6.2.15 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

5.6.2.16 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

5.6.2.17 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

5.6.2.18 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

5.6.2.19 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

5.6.2.20 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

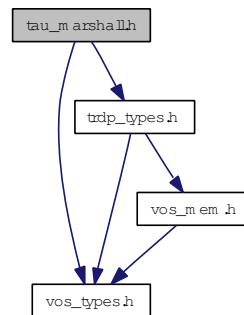
5.7 tau_marshall.h File Reference

TRDP utility interface definitions.

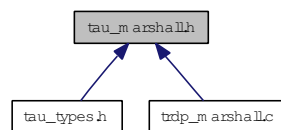
```
#include "vos_types.h"
```

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

Include dependency graph for tau_marshall.h:



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



Typedefs

- typedef [TRDP_ERR_T](#) [tau_marshallIDs](#) (void *pRefCon, UINT32 datasetId, const UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)
Marshall data set function.
- typedef [TRDP_ERR_T](#) [tau_unmarshallIDs](#) (void *pRefCon, UINT32 datasetId, const UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)
unmarshall data set function.
- typedef [TRDP_ERR_T](#) [tau_calcDatasetSize](#) (void *pRefCon, UINT32 datasetId, UINT8 *pSrc, UINT32 *pSize)
Calculate data set size.

Functions

- EXT_DECL [TRDP_ERR_T](#) [tau_initMarshall](#) (void **ppRefCon, UINT32 numComId, [TRDP_COMID_DSID_MAP_T](#) *pComIdsIdMap, 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)
marshall function.
- EXT_DECL [TRDP_ERR_T tau_unmarshall](#) (void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)
unmarshall function.

5.7.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- marshall/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](#) 94 2012-10-31 13:51:06Z 97025

5.7.2 Typedef Documentation

5.7.2.1 typedef TRDP_ERR_T tau_calcDatasetSize(void *pRefCon, UINT32 datasetId, UINT8 *pSrc, UINT32 *pSize)

Calculate data set size.

Parameters:

- ← *pRefCon* Pointer to user context
- ← *datasetId* Dataset id to identify the structure out of a configuration
- ← *pSrc* Pointer to received original message
- *pSize* Pointer to the size of the data set

Return values:

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

5.7.2.2 typedef TRDP_ERR_T tau_marshallDs(void *pRefCon, UINT32 datasetId, const UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)

Marshall data set function.

Parameters:

- ← *pRefCon* pointer to user context
- ← *datasetId* Dataset 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

Return values:

- TRDP_NO_ERR* no error
- TRDP_MEM_ERR* provided buffer to small
- TRDP_INIT_ERR* marshalling not initialised
- TRDP_PARAM_ERR* data set id not existing

5.7.2.3 typedef TRDP_ERR_T tau_unmarshallDs(void *pRefCon, UINT32 datasetId, const UINT8 *pSrc, UINT8 *pDest, UINT32 *pDestSize)

unmarshall data set function.

Parameters:

- ← *pRefCon* pointer to user context
- ← *datasetId* Dataset 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

Return values:

- TRDP_NO_ERR* no error
- TRDP_MEM_ERR* provided buffer to small
- TRDP_INIT_ERR* marshalling not initialised
- TRDP_PARAM_ERR* data set id not existing

5.7.3 Function Documentation

5.7.3.1 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
- ← *pComIdsIdMap* 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

5.7.3.2 EXT_DECL TRDP_ERR_T tau_marshall (void * *pRefCon*, UINT32 *comId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*)

marshall function.

Parameters:

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

Return values:

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

5.7.3.3 EXT_DECL TRDP_ERR_T tau_unmarshall (void * *pRefCon*, UINT32 *comId*, UINT8 * *pSrc*, UINT8 * *pDest*, UINT32 * *pDestSize*)

unmarshall function.

Parameters:

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

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR provided buffer too small

TRDP_INIT_ERR marshalling not initialised

TRDP_COMID_ERR comid not existing

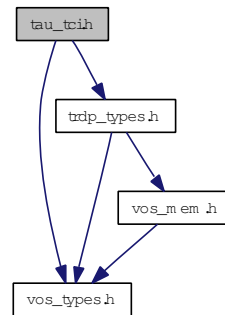
5.8 tau_tci.h File Reference

TRDP utility interface definitions.

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

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

Include dependency graph for tau_tci.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.

5.8.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- train configuration 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](#) 8 2012-06-06 16:28:19Z 97025

5.8.2 Enumeration Type Documentation

5.8.2.1 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.

5.8.2.2 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

5.8.3 Function Documentation

5.8.3.1 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

5.8.3.2 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

5.8.3.3 *****

**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

5.8.3.4 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

5.8.3.5 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

5.8.3.6 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

5.8.3.7 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

5.8.3.8 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

5.8.3.9 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

5.8.3.10 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

5.8.3.11 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

5.8.3.12 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

5.8.3.13 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:

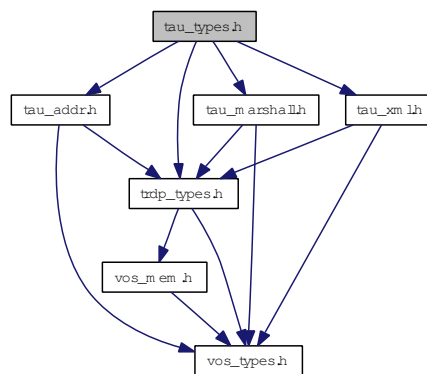
- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* Parameter error

5.9 tau_types.h File Reference

TRDP utility interface definitions.

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

Include dependency graph for tau_types.h:



5.9.1 Detailed Description

TRDP utility interface definitions.

This module provides the interface to the following utilities

- marshalling/unmarshall
- xml configuration interpreter
- 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_types.h](#) 2 2012-06-04 11:25:16Z 97025

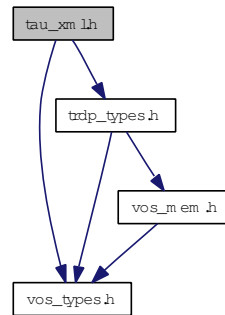
5.10 tau_xml.h File Reference

TRDP utility interface definitions.

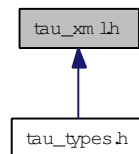
```
#include "vos_types.h"
```

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

Include dependency graph for tau_xml.h:



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



Data Structures

- struct [TRDP_PROCESS_CONFIG_T](#)
Types to read out the XML configuration.
- struct [TRDP_DBG_CONFIG_T](#)
Control for debug output device/file on application level.

Enumerations

- enum [TRDP_DBG_OPTION_T](#) {
[TRDP_DBG_DEFAULT](#) = 0,
[TRDP_DBG_OFF](#) = 0x01,
[TRDP_DBG_ERR](#) = 0x02,
[TRDP_DBG_WARN](#) = 0x04,
[TRDP_DBG_INFO](#) = 0x08,
[TRDP_DBG_DBG](#) = 0x10,
[TRDP_DBG_TIME](#) = 0x20,

```
TRDP_DBG_LOC = 0x40,
TRDP_DBG_CAT = 0x80 }
```

Control for debug output format on application level.

Functions

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

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

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

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

5.10.1 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](#) 2 2012-06-04 11:25:16Z 97025

5.10.2 Enumeration Type Documentation

5.10.2.1 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).

5.10.3 Function Documentation

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

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

Parameters:

← *pFileName* Path and filename of the xml configuration file
→ *pProcessConfig* TRDP main process configuration
→ *pMemConfig* Memory configuration
→ *pPdConfig* PD default configuration
→ *pMdConfig* MD default configuration
→ *pNumExchgPar* Number of configured telegrams
→ *ppExchgPar* Pointer to array of telegram configurations
→ *pNumComPar* Number of configured com parameters
→ *ppComPar* Pointer to array of com parameters
→ *pDbgPar* Debug printout options for application use

Return values:

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

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

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

Parameters:

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

- *pNumDataset* Pointer to the number of datasets found in the configuration
- *ppDataset* Pointer to an array of a structures of type [TRDP_DATASET_T](#)

Return values:

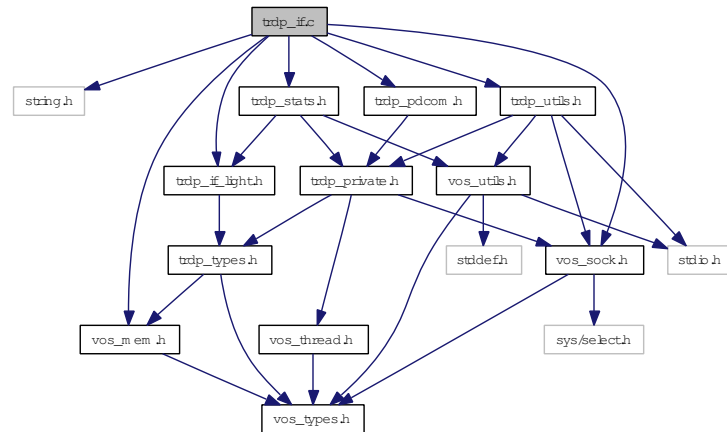
- TRDP_NO_ERR* no error
- TRDP_MEM_ERR* provided buffer to small
- TRDP_PARAM_ERR* File not existing

5.11 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_getVersion](#) (void)
Return a human readable version representation.
- [TRDP_ERR_T tlp_setRedundant](#) (TRDP_APP_SESSION_T appHandle, UINT32 redId, BOOL leader)
Do not send non-redundant PDs when we are follower.
- EXT_DECL [TRDP_ERR_T tlp_getRedundant](#) (TRDP_APP_SESSION_T appHandle, UINT32 redId, BOOL *pLeader)
Get status of redundant ComIds.
- EXT_DECL [TRDP_ERR_T tlc_setTopoCount](#) (TRDP_APP_SESSION_T appHandle, UINT32 topoCount)
Set new topocount for trainwide communication.
- UINT32 [trdp_getTopoCount](#) (TRDP_APP_SESSION_T appHandle)
Get current topocount.
- EXT_DECL [TRDP_ERR_T tlp_publish](#) (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T *pPubHandle, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, UINT32 interval, UINT32 redId, TRDP_FLAGS_T pktFlags, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, BOOL subs, UINT16 offsetAddress)
Prepare for sending PD messages.
- [TRDP_ERR_T tlp_unpublish](#) (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T pubHandle)
Stop sending PD messages.
- [TRDP_ERR_T tlp_put](#) (TRDP_APP_SESSION_T appHandle, TRDP_PUB_T pubHandle, const UINT8 *pData, UINT32 dataSize)
Update the process data to send.
- EXT_DECL [TRDP_ERR_T tlc_getInterval](#) (TRDP_APP_SESSION_T appHandle, TRDP_TIME_T *pInterval, TRDP_FDS_T *pFileDesc, INT32 *pNoDesc)
Get the lowest time interval for PDs.
- EXT_DECL [TRDP_ERR_T tlc_process](#) (TRDP_APP_SESSION_T appHandle, TRDP_FDS_T *pRfds, INT32 *pCount)
Work loop of the TRDP handler.
- EXT_DECL [TRDP_ERR_T tlp_request](#) (TRDP_APP_SESSION_T appHandle, TRDP_SUB_T subHandle, UINT32 comId, UINT32 topoCount, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, UINT32 redId, TRDP_FLAGS_T pktFlags, const TRDP_SEND_PARAM_T *pSendParam, const UINT8 *pData, UINT32 dataSize, UINT32 replyComId, TRDP_IP_ADDR_T replyIpAddr, BOOL subs, UINT16 offsetAddr)

Initiate sending PD messages (PULL).

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

Prepare for receiving PD messages.

- EXT_DECL [TRDP_ERR_T](#) [tlp_unsubscribe](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_SUB_T](#) subHandle)

Stop receiving PD messages.

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

Get the last valid PD message.

- [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.

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

Initiate sending MD request message.

- [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](#) destIpAddr, [TRDP_FLAGS_T](#) pktFlags, const [TRDP_URI_USER_T](#) destURI)

Subscribe to MD messages.

- [TRDP_ERR_T](#) [tlm_delListener](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_LIS_T](#) listenHandle)

Remove Listener.

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

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

- [TRDP_ERR_T tlm_replyErr](#) ([TRDP_APP_SESSION_T](#) appHandle, [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 reply message.

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

Initiate sending MD confirm message.

5.11.1 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](#) 159 2012-11-20 16:51:12Z bloehr

5.11.2 Function Documentation

5.11.2.1 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

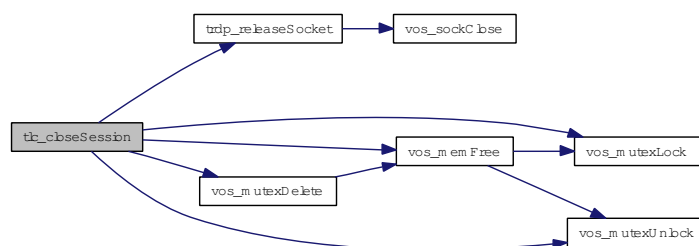
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Here is the call graph for this function:



5.11.2.2 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

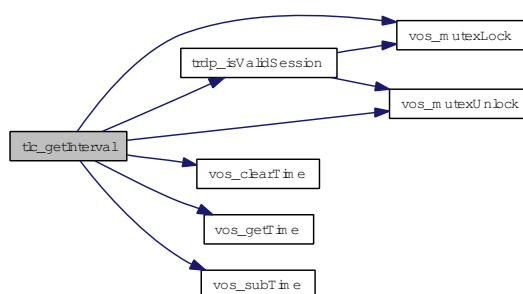
Parameters:

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

Return values:

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

Here is the call graph for this function:



5.11.2.3 const char* tlc_getVersion (void)

Return a human readable version representation.

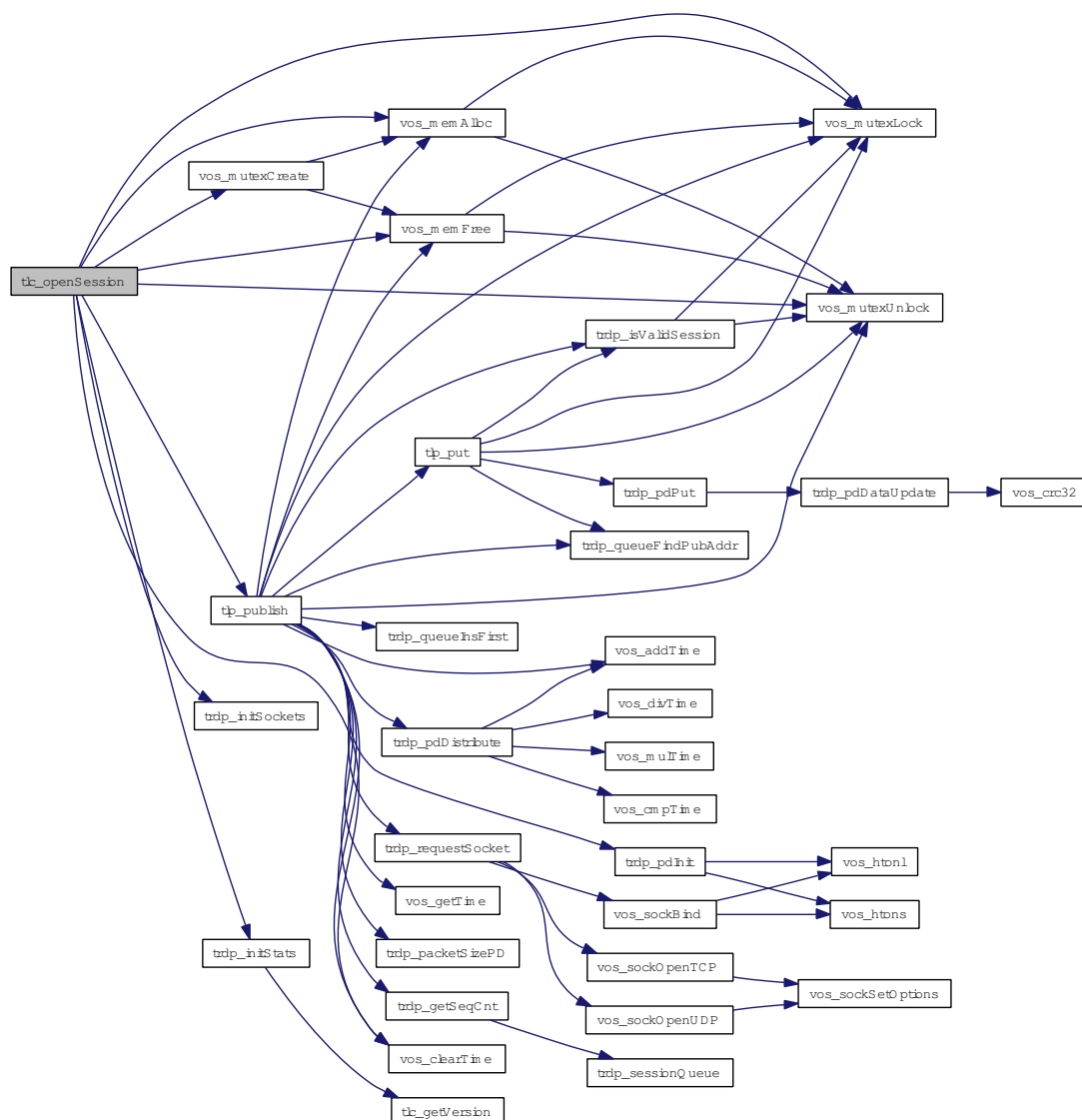
Return string in the form 'v.r.u.b'

- ← *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 initd
- TRDP_PARAM_ERR** parameter error
- TRDP SOCK_ERR** socket error

Here is the call graph for this function:



5.11.2.6 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

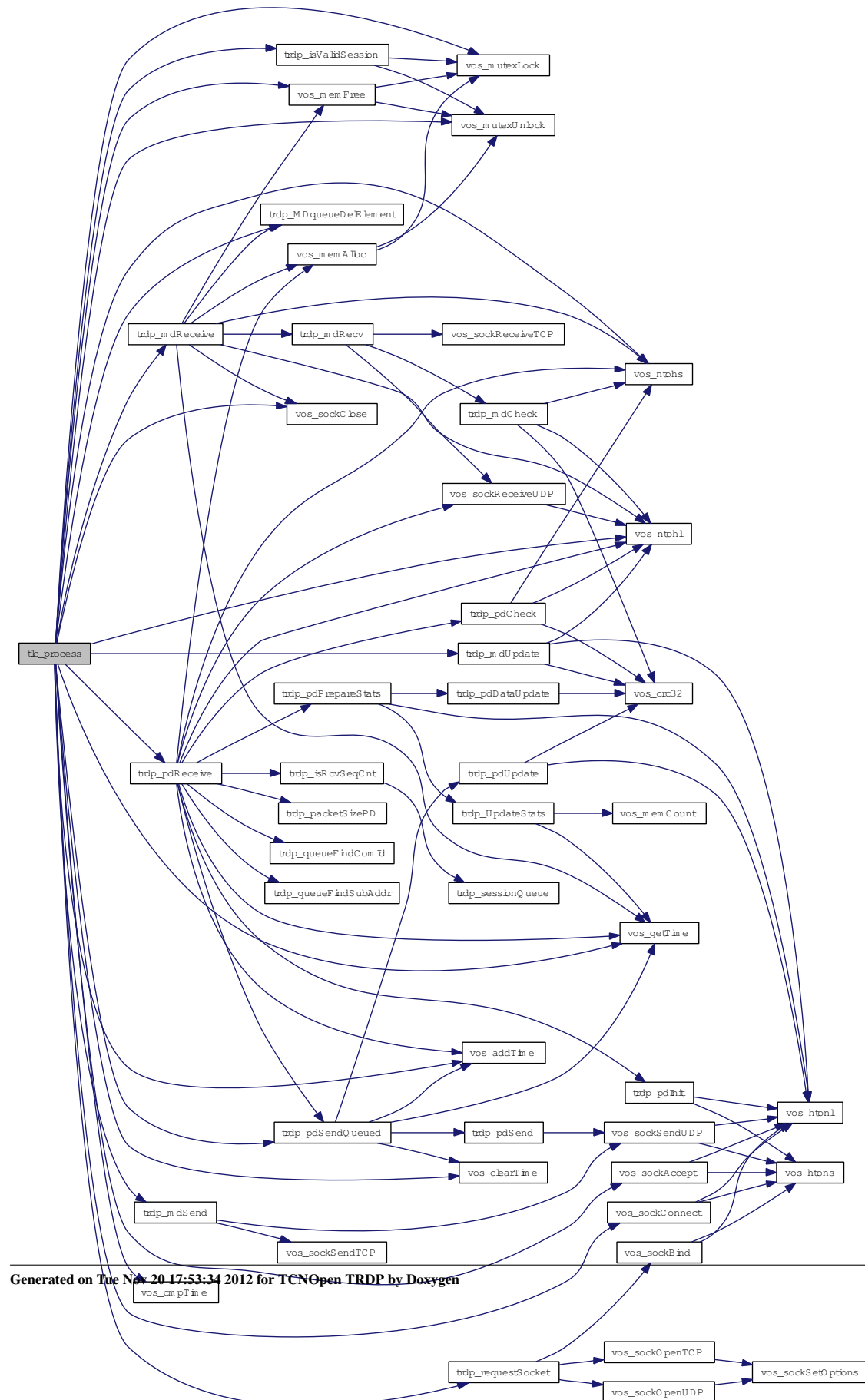
↔ *pCount* pointer to number of ready descriptors

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.11.2.7 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

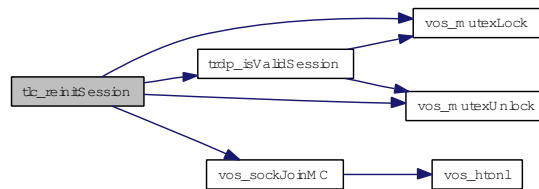
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Here is the call graph for this function:



5.11.2.8 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

Here is the call graph for this function:



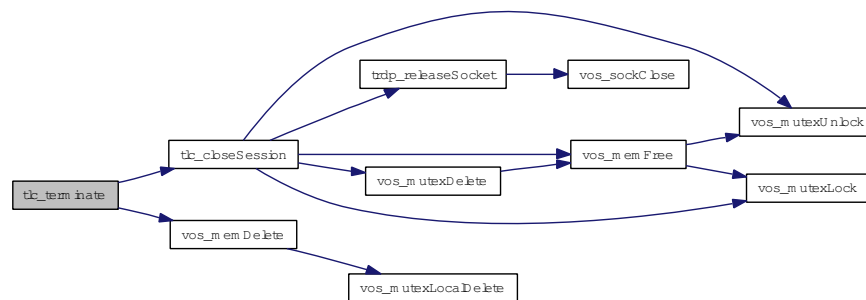
5.11.2.9 EXT_DECL TRDP_ERR_T tlc_terminate (void)

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

Here is the call graph for this function:



5.11.2.10 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 *destIpAddr*, TRDP_FLAGS_T *pktFlags*, const TRDP_URI_USER_T *destURI*)

Subscribe to MD messages.

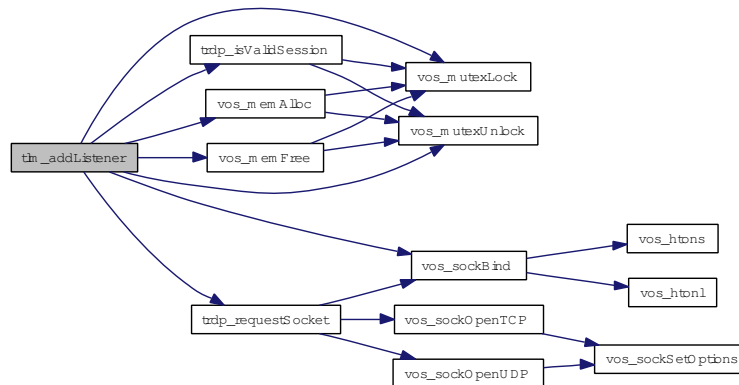
Add a listener to TRDP to get notified when messages are received

Parameters:

- ← ***appHandle*** the handle returned by tlc_init
- ***pListenHandle*** Listener ID returned
- ← ***pUserRef*** user supplied value returned with reply
- ← ***comId*** comId to be observed
- ← ***topoCount*** topocount to use
- ← ***destIpAddr*** destination IP address
- ← ***pktFlags*** optional marshalling
- ← ***destURI*** only functional group of destination URI

Return values:**TRDP_NO_ERR** no error**TRDP_PARAM_ERR** parameter error**TRDP_MEM_ERR** out of memory**TRDP_NOINIT_ERR** handle invalid

Here is the call graph for this function:



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

Initiate sending MD confirm message.

Send a MD confirmation message

Parameters:

- ← *appHandle* the handle returned by *tlc_init*
- ← *pUserRef* user supplied value returned with reply
- ← *pSessionId* Session ID returned by request
- ← *comId* comId of packet to be sent
- ← *topoCount* topocount to use
- ← *srcIpAddr* own IP address, 0 - srcIP will be set by the stack
- ← *destIpAddr* where to send the packet to
- ← *pktFlags* OPTION: **TRDP_FLAGS_CALLBACK**
- ← *userStatus* Info for requester about application errors
- ← *replyStatus* Info for requester about stack errors
- ← *pSendParam* Pointer to send parameters, NULL to use default send parameters
- ← *srcURI* only functional group of source URI
- ← *destURI* only functional group of destination URI

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_MEM_ERR** out of memory
- TRDP_NO_SESSION_ERR** no such session
- TRDP_NOINIT_ERR** handle invalid

5.11.2.12 TRDP_ERR_T tlm_delListener (TRDP_APP_SESSION_T *appHandle*, TRDP_LIS_T *listenHandle*)

Remove Listener.

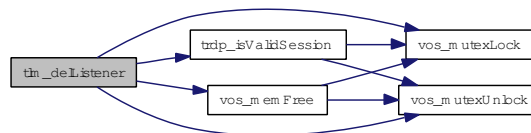
Parameters:

- ← *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

Here is the call graph for this function:



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

Initiate sending MD notification message.

Send a MD notification message

Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *pUserRef* user supplied value returned with reply
- ← *comId* comId of packet to be sent
- ← *topoCount* topocount to use
- ← *srcIpAddr* own IP address, 0 - srcIP will be set by the stack
- ← *destIpAddr* where to send the packet to
- ← *pktFlags* OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *pSendParam* optional pointer to send parameter, NULL - default parameters are used
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data
- ← *sourceURI* only functional group of source URI

← *destURI* only functional group of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NOINIT_ERR handle invalid

5.11.2.14 *TRDP_ERR_T* *tlm_reply* (*TRDP_APP_SESSION_T* *appHandle*, void * *pUserRef*, *TRDP_UUID_T* * *pSessionId*, *UINT32* *topoCount*, *UINT32* *comId*, *TRDP_IP_ADDR_T* *srcIpAddr*, *TRDP_IP_ADDR_T* *destIpAddr*, *TRDP_FLAGS_T* *pktFlags*, *UINT16* *userStatus*, 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* optional marshalling

← *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

5.11.2.15 `TRDP_ERR_T tlm_replyErr (TRDP_APP_SESSION_T appHandle, 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 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

5.11.2.16 `TRDP_ERR_T tlm_replyQuery (TRDP_APP_SESSION_T appHandle, void *pUserRef, TRDP_UUID_T *pSessionId, UINT32 topoCount, UINT32 comId, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, UINT16 userStatus, 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* optional marshalling
- ← *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

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

Initiate sending MD request message.

Send a MD request message

Parameters:

- ← *appHandle* the handle returned by *tlc_init*
- ← *pUserRef* user supplied value returned with reply
- *pSessionId* return session ID
- ← *comId* comId of packet to be sent
- ← *topoCount* topocount to use
- ← *srcIpAddr* own IP address, 0 - srcIP will be set by the stack
- ← *destIpAddr* where to send the packet to
- ← *pktFlags* OPTIONS: *TRDP_FLAGS_MARSHALL*, *TRDP_FLAGS_CALLBACK*
- ← *noOfRepliers* number of expected repliers, 0 if unknown
- ← *replyTimeout* timeout for reply
- ← *pSendParam* Pointer to send parameters, NULL to use default send parameters
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data

← *srcURI* only functional group of source URI

← *destURI* only functional group of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NOINIT_ERR handle invalid

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

Get the last valid PD message.

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

Parameters:

← *appHandle* the handle returned by `tlc_openSession`

← *subHandle* the handle returned by subscription

← *pktFlags* OPTION: **TRDP_FLAGS_MARSHALL**

↔ *pPdInfo* pointer to application's info buffer

↔ *pData* pointer to application's data buffer

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

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

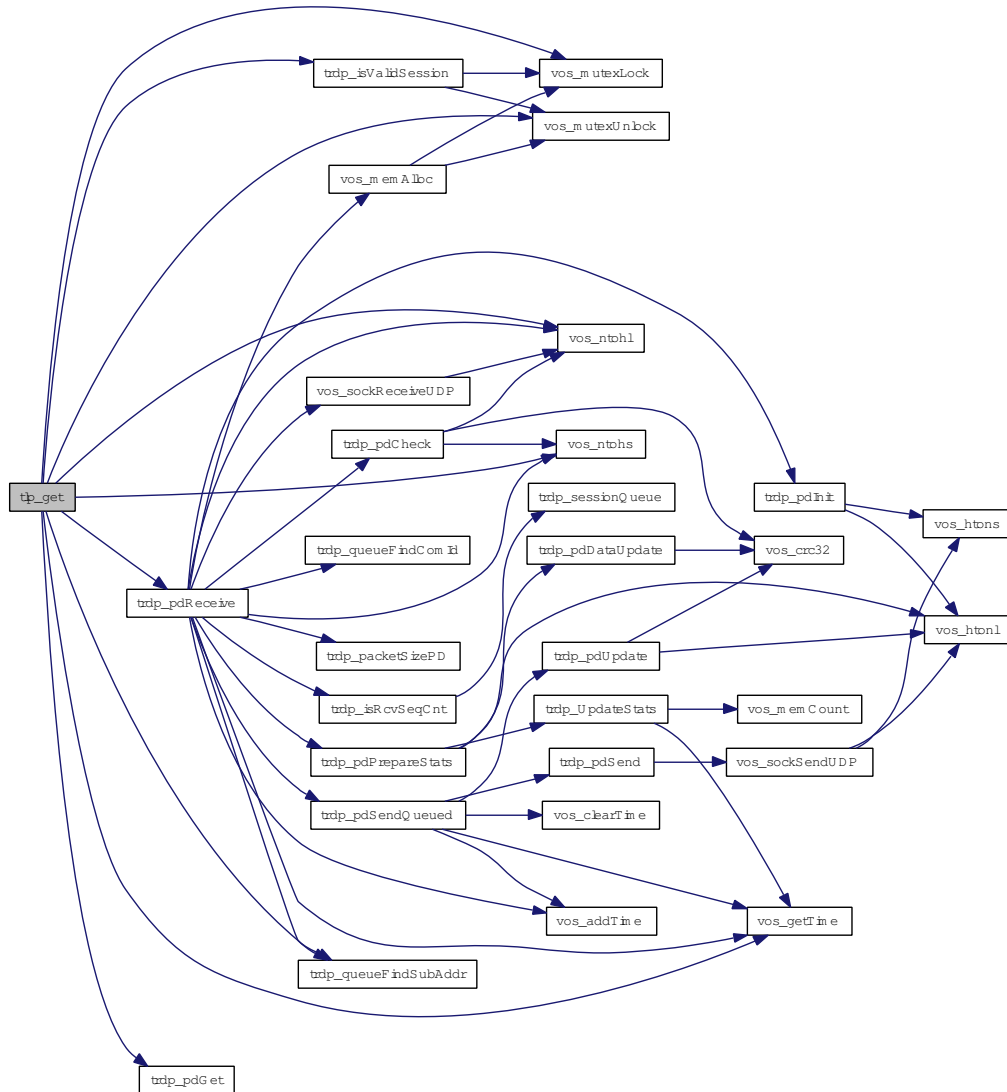
TRDP_SUB_ERR not subscribed

TRDP_TIMEOUT_ERR packet timed out

TRDP_NOINIT_ERR handle invalid

TRDP_COMID_ERR ComID not found when marshalling

Here is the call graph for this function:



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

Get status of redundant ComIds.

Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *redId* will be returned for all ComID's with the given redId, 0 for all redId
- ↔ *pLeader* TRUE if we send (leader)

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error / redId not existing

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



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

Prepare for sending PD messages.

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

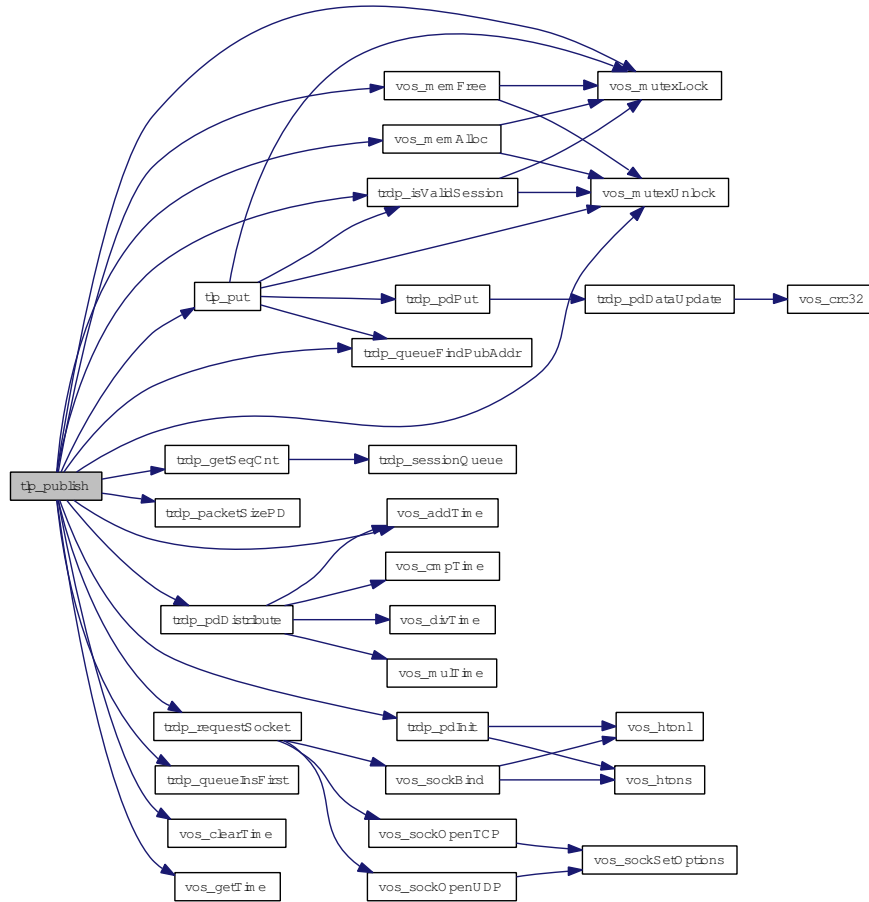
Parameters:

- ← *appHandle* the handle returned by tlc_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* OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *pSendParam* optional pointer to send parameter, NULL - default parameters are used
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data <= 1436 without FCS
- ← *subs* substitution (Ladder)
- ← *offsetAddress* offset (Ladder)

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_MEM_ERR** could not insert (out of memory)
- TRDP_NOINIT_ERR** handle invalid
- TRDP_NOPUB_ERR** Already published

Here is the call graph for this function:



5.11.2.21 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:

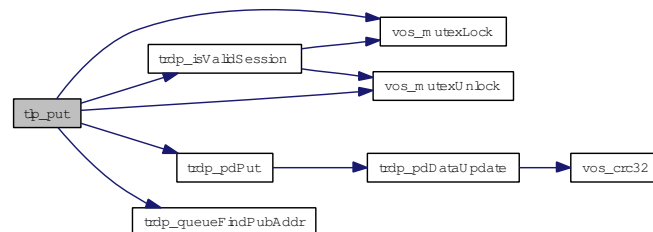
- ← **appHandle** the handle returned by tlc_openSession
- ← **pubHandle** the handle returned by publish
- ↔ **pData** pointer to application's data buffer
- ↔ **dataSize** size of data

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error

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:



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

Initiate sending PD messages (PULL).

Send a PD request message

Parameters:

- ← **appHandle** the handle returned by tlc_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** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← **pSendParam** optional pointer to send parameter, NULL - default parameters are used
- ← **pData** pointer to packet data / dataset
- ← **dataSize** size of packet data
- ← **replyComId** comId of reply
- ← **replyIpAddr** IP for reply
- ← **subs** substitution (Ladder)
- ← **offsetAddr** offset (Ladder)

Return values:

TRDP_NO_ERR no error

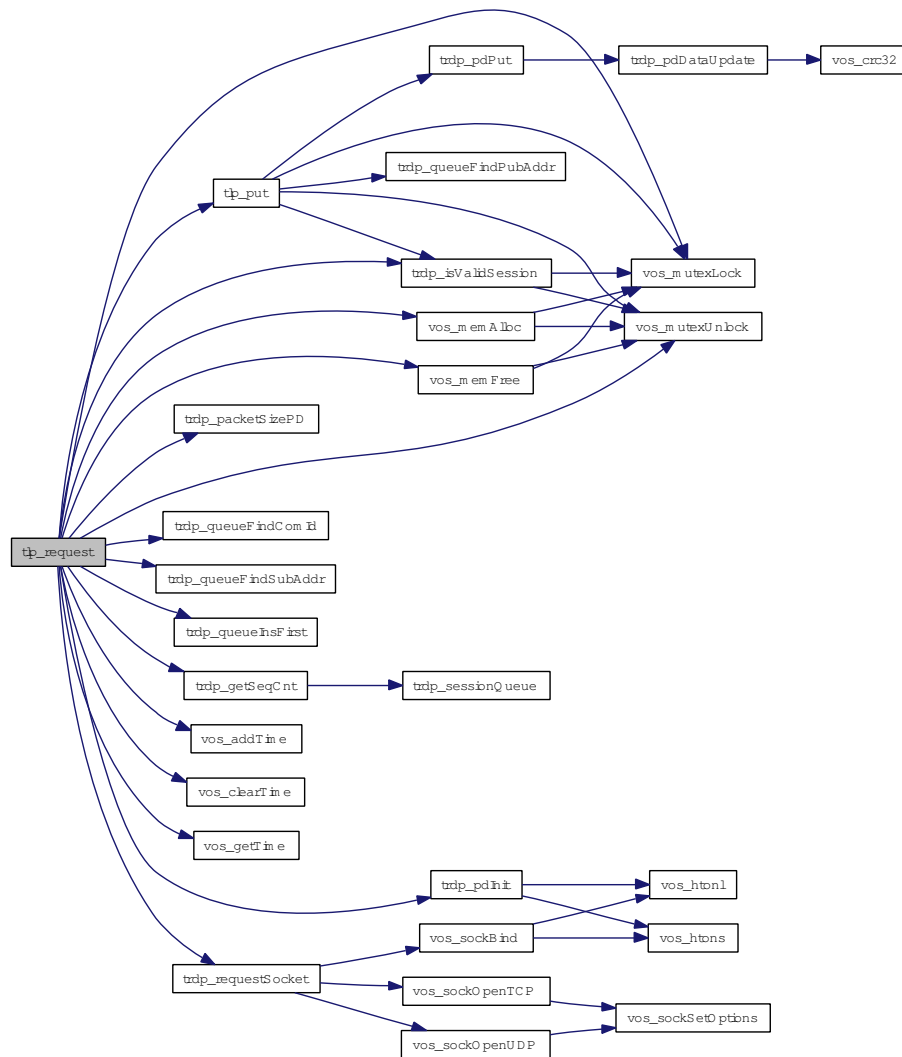
TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

TRDP_NOSUB_ERR no matching subscription found

Here is the call graph for this function:



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

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

Parameters:

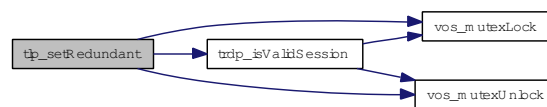
← *appHandle* the handle returned by `tlc_init`

- ← **redId** will be set for all ComID's with the given redId, 0 to change for all redId
- ← **leader** TRUE if we send

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error / redId not existing
- TRDP_NOINIT_ERR** handle invalid

Here is the call graph for this function:



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

Prepare for receiving PD messages.

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

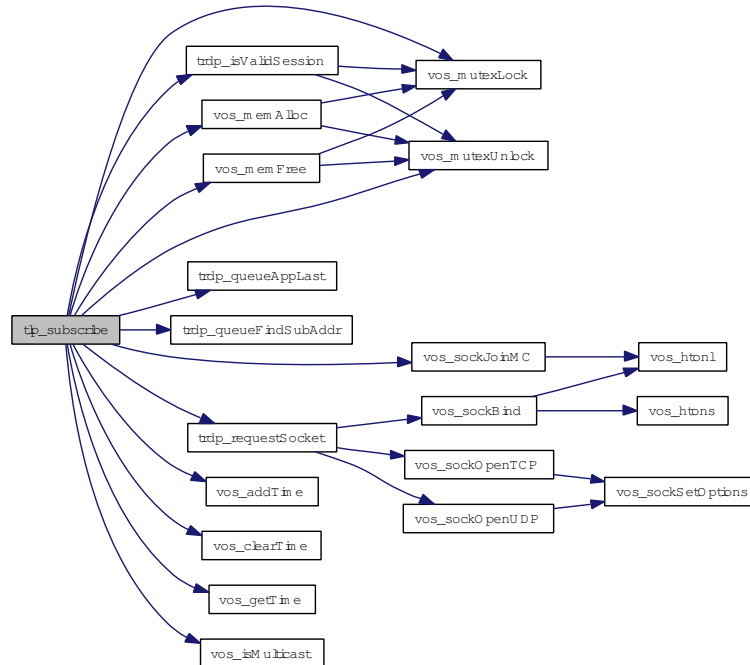
Parameters:

- ← **appHandle** the handle returned by tlc_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.
- ← **destIpAddr** IP address to join
- ← **timeout** timeout (>= 10ms) in usec
- ← **toBehavior** timeout behavior
- ← **maxDataSize** expected max. size of packet data

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_MEM_ERR** could not reserve memory (out of memory)
- TRDP_NOINIT_ERR** handle invalid

Here is the call graph for this function:



5.11.2.25 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:

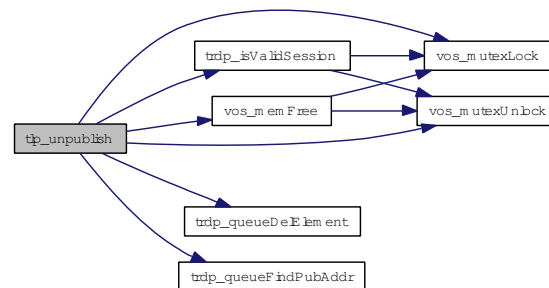
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOPUB_ERR not published

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.11.2.26 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

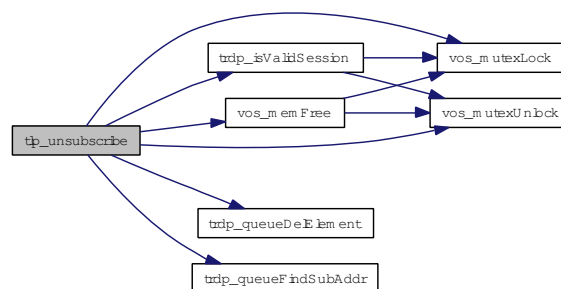
Parameters:

- ← *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_SUB_ERR** not subscribed
- TRDP_NOINIT_ERR** handle invalid

Here is the call graph for this function:



5.11.2.27 UINT32 trdp_getTopoCount (TRDP_APP_SESSION_T *appHandle*)

Get current topocount.

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

Return values:

topoCount Current topoCount value

5.11.2.28 `BOOL trdp_isValidSession (TRDP_APP_SESSION_T pSessionHandle)`

Check if the session handle is valid.

Parameters:

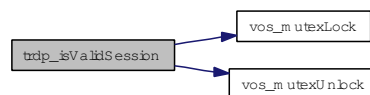
← *pSessionHandle* pointer to packet data (dataset)

Return values:

TRUE is valid

FALSE is invalid

Here is the call graph for this function:

**5.11.2.29** `TRDP_APP_SESSION_T* trdp_sessionQueue (void)`

Get the session queue head pointer.

Return values:

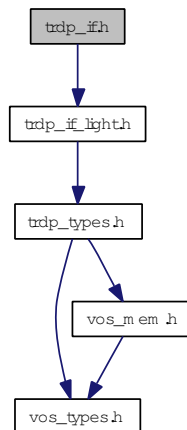
&sSession

5.12 trdp_if.h File Reference

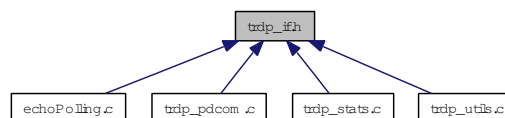
Typedefs for TRDP communication.

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

Include dependency graph for trdp_if.h:



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



Functions

- `UINT32 trdp_getTopoCount (void)`
Get current topocount.
- `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.

5.12.1 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](#) 53 2012-10-17 17:40:43Z 97025

5.12.2 Function Documentation**5.12.2.1 UINT32 trdp_getTopoCount (void)**

Get current topocount.

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

Return values:

topoCount Current topoCount value

5.12.2.2 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:

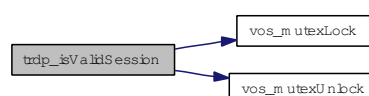
← *pSessionHandle* pointer to packet data (dataset)

Return values:

TRUE is valid

FALSE is invalid

Here is the call graph for this function:



5.12.2.3 TRDP_APP_SESSION_T* trdp_sessionQueue (void)

Get the session queue head pointer.

Return values:

&sSession

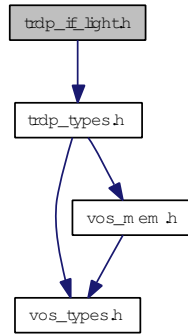
&sSession

5.13 trdp_if_light.h File Reference

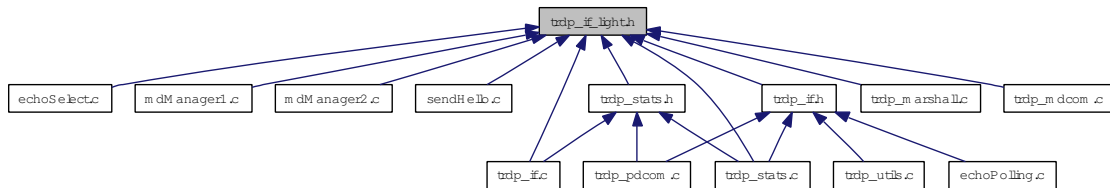
TRDP Light interface functions (API).

```
#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, [BOOL](#) subs, [UINT16](#) offsetAddress)
Prepare for sending PD messages.
- EXT_DECL [TRDP_ERR_T tlp_unpublish](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_PUB_T](#) pubHandle)
Stop sending PD messages.
- EXT_DECL [TRDP_ERR_T tlp_put](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_PUB_T](#) pubHandle, const [UINT8](#) *pData, [UINT32](#) dataSize)
Update the process data to send.
- EXT_DECL [TRDP_ERR_T tlp_setRedundant](#) ([TRDP_APP_SESSION_T](#) appHandle, [UINT32](#) redId, [BOOL](#) leader)
Do not send non-redundant PDs when we are follower.
- EXT_DECL [TRDP_ERR_T tlp_getRedundant](#) ([TRDP_APP_SESSION_T](#) appHandle, [UINT32](#) redId, [BOOL](#) *pLeader)
Get status of redundant ComIds.
- EXT_DECL [TRDP_ERR_T tlp_request](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_SUB_T](#) subHandle, [UINT32](#) comId, [UINT32](#) topoCount, [TRDP_IP_ADDR_T](#) srcIpAddr, [TRDP_IP_ADDR_T](#) destIpAddr, [UINT32](#) redId, [TRDP_FLAGS_T](#) pktFlags, const [TRDP_SEND_PARAM_T](#) *pSendParam, const [UINT8](#) *pData, [UINT32](#) dataSize, [UINT32](#) replyComId, [TRDP_IP_ADDR_T](#) replyIpAddr, [BOOL](#) subs, [UINT16](#) offsetAddr)
Initiate sending PD messages (PULL).

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

Prepare for receiving PD messages.

- EXT_DECL [TRDP_ERR_T tlp_unsubscribe](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_SUB_T](#) subHandle)

Stop receiving PD messages.

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

Get the last valid PD message.

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

Initiate sending MD notification message.

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

Initiate sending MD request message.

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

Initiate sending MD confirm message.

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

Cancel an open session.

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

Subscribe to MD messages.

- EXT_DECL [TRDP_ERR_T tlm_delListener](#) ([TRDP_APP_SESSION_T](#) appHandle, [TRDP_LIS_T](#) listenHandle)

Remove Listener.

- EXT_DECL [TRDP_ERR_T tlm_reply](#) ([TRDP_APP_SESSION_T](#) appHandle, void *pUserRef, [TRDP_UUID_T](#) *pSessionId, UINT32 topoCount, UINT32 comId, [TRDP_IP_ADDR_T](#) srcIpAddr,

[TRDP_IP_ADDR_T](#) destIpAddr, [TRDP_FLAGS_T](#) pktFlags, UINT16 userStatus, const [TRDP_SEND_PARAM_T](#) *pSendParam, const UINT8 *pData, UINT32 dataSize, const [TRDP_URI_USER_T](#) srcURI, const [TRDP_URI_USER_T](#) destURI)

Send a MD reply message.

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

Send a MD reply message.

- EXT_DECL [TRDP_ERR_T](#) [tlm_replyErr](#) ([TRDP_APP_SESSION_T](#) appHandle, [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](#) srcURI, const [TRDP_URI_USER_T](#) destURI)

Send a MD reply message.

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

Return a human readable version representation.

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

5.13.1 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](#) 131 2012-11-07 14:03:09Z 97025

5.13.2 Function Documentation

5.13.2.1 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

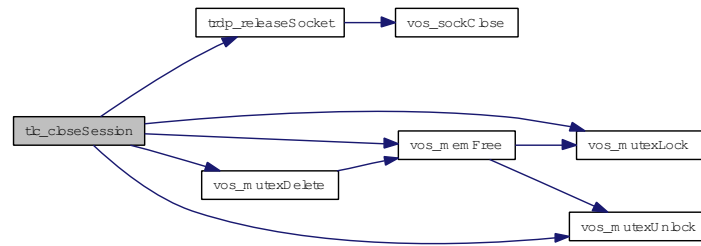
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Here is the call graph for this function:



5.13.2.2 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

5.13.2.3 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

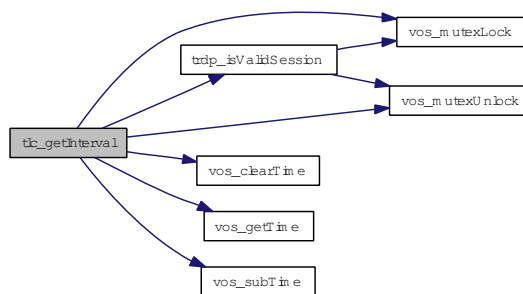
Parameters:

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

Return values:

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

Here is the call graph for this function:



5.13.2.4 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 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

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 Addresses

→ *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:



5.13.2.5 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:

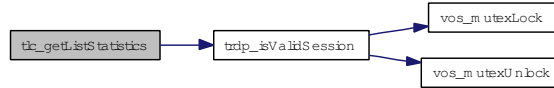
TRDP_NO_ERR no error

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:



5.13.2.6 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:

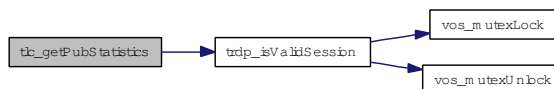
TRDP_NO_ERR no error

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:



5.13.2.7 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.

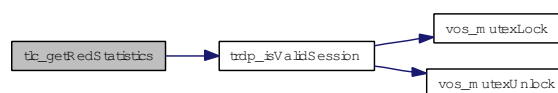
Parameters:

- ← *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:



5.13.2.8 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.

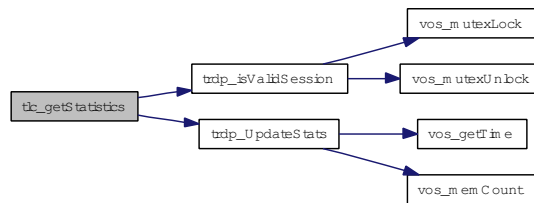
Parameters:

- ← *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:



5.13.2.9 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.

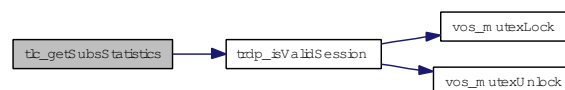
Parameters:

← **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:



5.13.2.10 EXT_DECL const CHAR8* tlc_getVersion (void)

Return a human readable version representation.

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

Return values:

const string

5.13.2.11 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

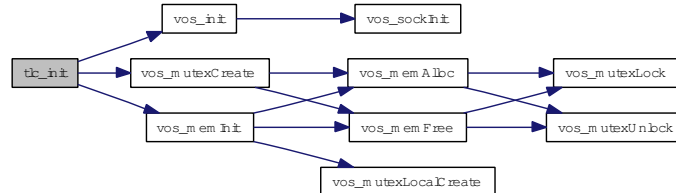
Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR memory allocation failed

TRDP_PARAM_ERR initialization error

Here is the call graph for this function:



5.13.2.12 **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

↔ *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

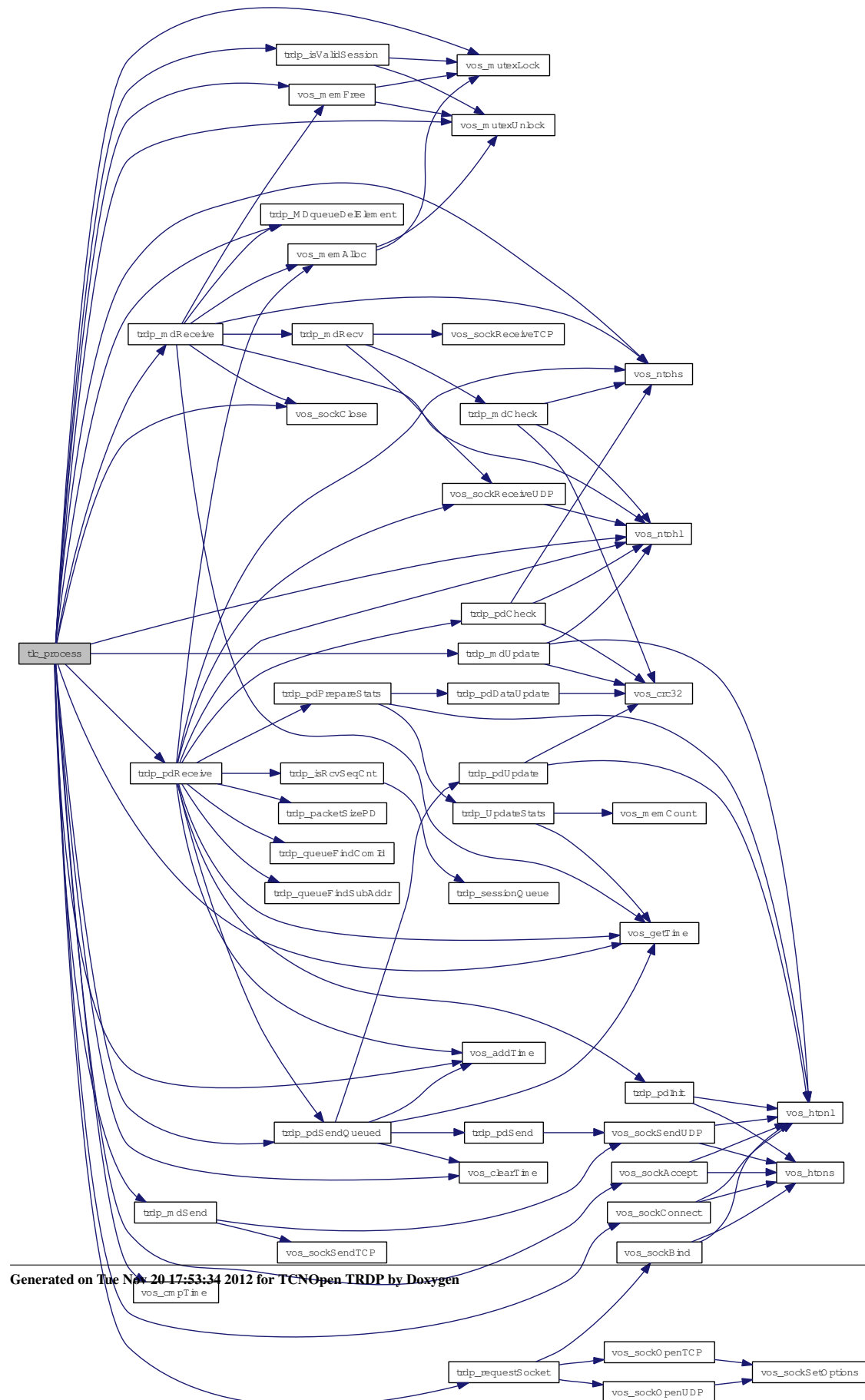
↔ *pCount* pointer to number of ready descriptors

Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



5.13.2.14 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

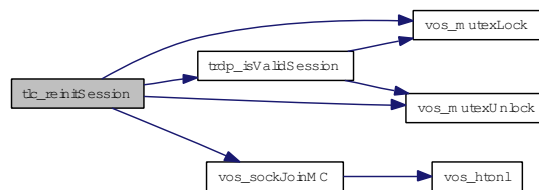
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR handle NULL

Here is the call graph for this function:



5.13.2.15 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`

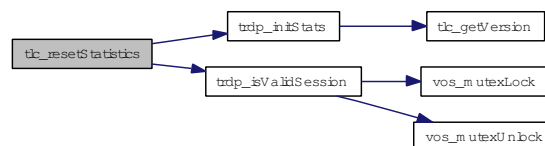
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.13.2.16 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

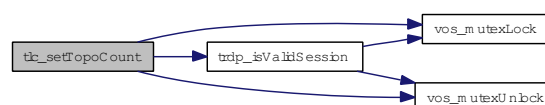
This value is used for validating outgoing and incoming packets only!

Parameters:

← *appHandle* the handle returned by `tlc_openSession`

← *topoCount* New topoCount value

Here is the call graph for this function:



5.13.2.17 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

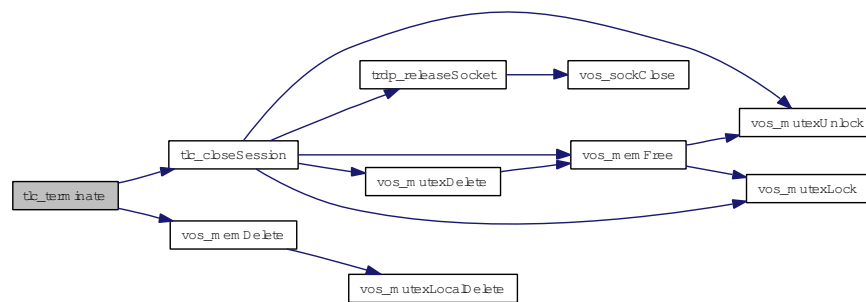
Clean up and close all sessions. Mainly used for debugging/test runs. No further calls to library allowed

Return values:

TRDP_NO_ERR no error

TRDP_INIT_ERR no error

Here is the call graph for this function:



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

Cancel an open session.

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

Parameters:

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

↔ **pSessionId** Session ID returned by request

Return values:

TRDP_NO_ERR no error

TRDP_NO_SESSION_ERR no such session

TRDP_NOINIT_ERR handle invalid

5.13.2.19 `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 destIpAddr, TRDP_FLAGS_T pktFlags, const TRDP_URI_USER_T destURI)`

Subscribe to MD messages.

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

Parameters:

- ← *appHandle* the handle returned by `tlc_init`
- *pListenHandle* Listener ID returned
- ← *pUserRef* user supplied value returned with reply
- ← *comId* comId to be observed
- ← *topoCount* topocount to use
- ← *destIpAddr* destination IP address
- ← *pktFlags* optional marshalling
- ← *destURI* only functional group of destination URI

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* parameter error
- TRDP_MEM_ERR* out of memory
- TRDP_NOINIT_ERR* handle invalid

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

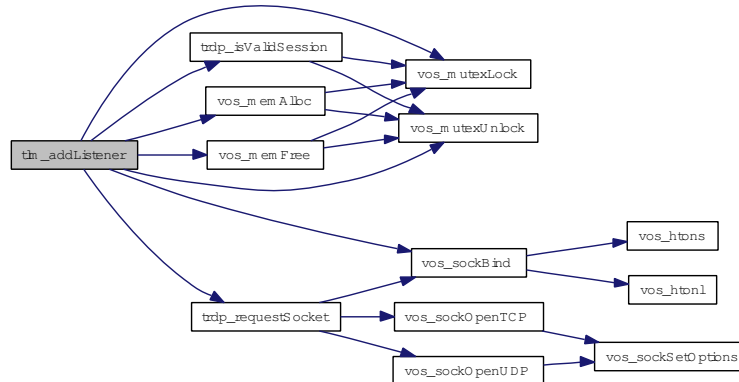
Parameters:

- ← *appHandle* the handle returned by `tlc_init`
- *pListenHandle* Listener ID returned
- ← *pUserRef* user supplied value returned with reply
- ← *comId* comId to be observed
- ← *topoCount* topocount to use
- ← *destIpAddr* destination IP address
- ← *pktFlags* optional marshalling
- ← *destURI* only functional group of destination URI

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* parameter error
- TRDP_MEM_ERR* out of memory
- TRDP_NOINIT_ERR* handle invalid

Here is the call graph for this function:



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

Initiate sending MD confirm message.

Send a MD confirmation message

Parameters:

- ← **appHandle** the handle returned by tlc_init
- ← **pUserRef** user supplied value returned with reply
- ← **pSessionId** Session ID returned by request
- ← **comId** comId of packet to be sent
- ← **topoCount** topocount to use
- ← **srcIpAddr** own IP address, 0 - srcIP will be set by the stack
- ← **destIpAddr** where to send the packet to
- ← **pktFlags** OPTION: TRDP_FLAGS_CALLBACK
- ← **userStatus** Info for requester about application errors
- ← **replyStatus** Info for requester about stack errors
- ← **pSendParam** Pointer to send parameters, NULL to use default send parameters
- ← **srcURI** only functional group of source URI
- ← **destURI** only functional group of destination URI

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_MEM_ERR** out of memory

TRDP_NO_SESSION_ERR no such session

TRDP_NOINIT_ERR handle invalid

Send a MD confirmation message

Parameters:

- ← **appHandle** the handle returned by tlc_init
- ← **pUserRef** user supplied value returned with reply
- ← **pSessionId** Session ID returned by request
- ← **comId** comId of packet to be sent
- ← **topoCount** topocount to use
- ← **srcIpAddr** own IP address, 0 - srcIP will be set by the stack
- ← **destIpAddr** where to send the packet to
- ← **pktFlags** OPTION: TRDP_FLAGS_CALLBACK
- ← **userStatus** Info for requester about application errors
- ← **replyStatus** Info for requester about stack errors
- ← **pSendParam** Pointer to send parameters, NULL to use default send parameters
- ← **srcURI** only functional group of source URI
- ← **destURI** only functional group of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NO_SESSION_ERR no such session

TRDP_NOINIT_ERR handle invalid

5.13.2.21 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

Parameters:

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

→ *listenHandle* Listener ID returned

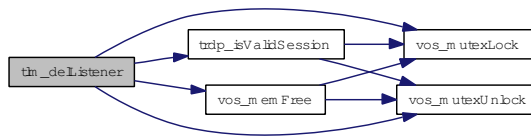
Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



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

Initiate sending MD notification message.

Send a MD notification message

Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *pUserRef* user supplied value returned with reply
- ← *comId* comId of packet to be sent
- ← *topoCount* topocount to use
- ← *srcIpAddr* own IP address, 0 - srcIP will be set by the stack
- ← *destIpAddr* where to send the packet to
- ← *pktFlags* OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *pSendParam* optional pointer to send parameter, NULL - default parameters are used
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data
- ← *sourceURI* only functional group of source URI
- ← *destURI* only functional group of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NOINIT_ERR handle invalid

Send a MD notification message

Parameters:

- ← **appHandle** the handle returned by `tlc_init`
- ← **pUserRef** user supplied value returned with reply
- ← **comId** comId of packet to be sent
- ← **topoCount** topocount to use
- ← **srcIpAddr** own IP address, 0 - srcIP will be set by the stack
- ← **destIpAddr** where to send the packet to
- ← **pktFlags** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← **pSendParam** optional pointer to send parameter, NULL - default parameters are used
- ← **pData** pointer to packet data / dataset
- ← **dataSize** size of packet data
- ← **sourceURI** only functional group of source URI
- ← **destURI** only functional group of destination URI

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_MEM_ERR** out of memory
- TRDP_NOINIT_ERR** handle invalid

5.13.2.23 `EXT_DECL TRDP_ERR_T tlm_reply (TRDP_APP_SESSION_T appHandle, void * pUserRef, TRDP_UUID_T * pSessionId, UINT32 topoCount, UINT32 comId, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, UINT16 userStatus, 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** optional marshalling

- ← *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
- ← *srcURI* only user part of source URI
- ← *destURI* only user part of destination URI

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* parameter error
- TRDP_MEM_ERR* out of memory
- TRDP_NO_SESSION_ERR* no such session
- TRDP_NOINIT_ERR* handle invalid

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* optional marshallng
- ← *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

5.13.2.24 **EXT_DECL** **TRDP_ERR_T** **tlm_replyErr** (**TRDP_APP_SESSION_T** *appHandle*, **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 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
- ← *srcURI* only user part of source URI
- ← *destURI* only user part of destination URI

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_MEM_ERR** out of memory
- TRDP_NO_SESSION_ERR** no such session
- TRDP_NOINIT_ERR** handle invalid

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

5.13.2.25 EXT_DECL TRDP_ERR_T tlm_replyQuery (TRDP_APP_SESSION_T appHandle, void * pUserRef, TRDP_UUID_T * pSessionId, UINT32 topoCount, UINT32 comId, TRDP_IP_ADDR_T srcIpAddr, TRDP_IP_ADDR_T destIpAddr, TRDP_FLAGS_T pktFlags, UINT16 userStatus, 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** optional marshalling
- ← **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
- ← **srcURI** only user part of source URI
- ← **destURI** only user part of destination URI

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR out of memory

TRDP_NO_SESSION_ERR no such session

TRDP_NOINIT_ERR handle invalid

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* optional marshalling
- ← *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

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

Initiate sending MD request message.

Send a MD request message

Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *pUserRef* user supplied value returned with reply
- *pSessionId* return session ID
- ← *comId* comId of packet to be sent
- ← *topoCount* topocount to use
- ← *srcIpAddr* own IP address, 0 - srcIP will be set by the stack
- ← *destIpAddr* where to send the packet to
- ← *pktFlags* OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *noOfRepliers* number of expected repliers, 0 if unknown

- ← *replyTimeout* timeout for reply
- ← *noOfRetries* number of retries
- ← *pSendParam* Pointer to send parameters, NULL to use default send parameters
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data
- ← *srcURI* only functional group of source URI
- ← *destURI* only functional group of destination URI

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* parameter error
- TRDP_MEM_ERR* out of memory
- TRDP_NOINIT_ERR* handle invalid

Send a MD request message

Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *pUserRef* user supplied value returned with reply
- *pSessionId* return session ID
- ← *comId* comId of packet to be sent
- ← *topoCount* topocount to use
- ← *srcIpAddr* own IP address, 0 - srcIP will be set by the stack
- ← *destIpAddr* where to send the packet to
- ← *pktFlags* OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *noOfRepliers* number of expected repliers, 0 if unknown
- ← *replyTimeout* timeout for reply
- ← *pSendParam* Pointer to send parameters, NULL to use default send parameters
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data
- ← *srcURI* only functional group of source URI
- ← *destURI* only functional group of destination URI

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* parameter error
- TRDP_MEM_ERR* out of memory
- TRDP_NOINIT_ERR* handle invalid

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

Get the last valid PD message.

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

Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *subHandle* the handle returned by subscription
- ← *pktFlags* OPTION: TRDP_FLAGS_MARSHALL
- ↔ *pPdInfo* pointer to application's info buffer
- ↔ *pData* pointer to application's data buffer
- ↔ *pDataSize* in: size of buffer, out: size of data

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_SUB_ERR** not subscribed
- TRDP_TIMEOUT_ERR** packet timed out
- TRDP_NOINIT_ERR** handle invalid
- 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
- ← *pktFlags* OPTION: TRDP_FLAGS_MARSHALL
- ↔ *pPdInfo* pointer to application's info buffer
- ↔ *pData* pointer to application's data buffer
- ↔ *pDataSize* in: size of buffer, out: size of data

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_SUB_ERR** not subscribed
- TRDP_TIMEOUT_ERR** packet timed out
- TRDP_NOINIT_ERR** handle invalid
- TRDP_COMID_ERR** ComID not found when marshalling

TRDP_PARAM_ERR parameter error / redId not existing

TRDP_NOINIT_ERR handle invalid

Parameters:

← **appHandle** the handle returned by tlc_init

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

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

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error / redId not existing

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



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

Prepare for sending PD messages.

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

Parameters:

← **appHandle** the handle returned by tlc_init

→ **pPubHandle** returned handle for related unprepare

← **comId** comId of packet to send

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

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

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

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

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

← **pktFlags** OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK

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

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

← **dataSize** size of packet data

← *subs* substitution (Ladder)

← *offsetAddress* offset (Ladder)

Return values:

TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

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

Parameters:

← *appHandle* the handle returned by tlc_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* OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK

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

← *pData* pointer to packet data / dataset

← *dataSize* size of packet data ≤ 1436 without FCS

← *subs* substitution (Ladder)

← *offsetAddress* offset (Ladder)

Return values:

TRDP_NO_ERR no error

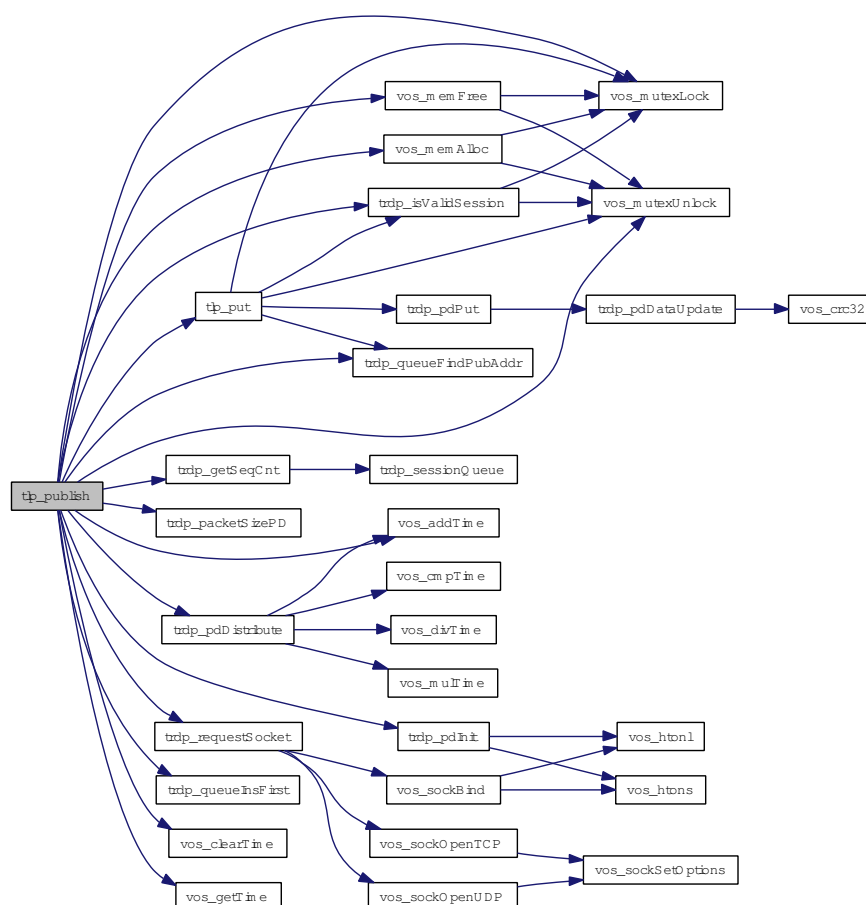
TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

TRDP_NOPUB_ERR Already published

Here is the call graph for this function:



5.13.2.30 EXT_DECL TRDP_ERR_T trdp_put (TRDP_APP_SESSION_T *appHandle*, TRDP_PUB_T *pubHandle*, const UINT8 * *pData*, UINT32 *dataSize*)

Update the process data to send.

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

Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *pubHandle* the handle returned by publish
- ↔ *pData* pointer to application's data buffer
- ↔ *dataSize* size of data

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error

TRDP_PUB_ERR not published

TRDP_NOINIT_ERR handle invalid

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

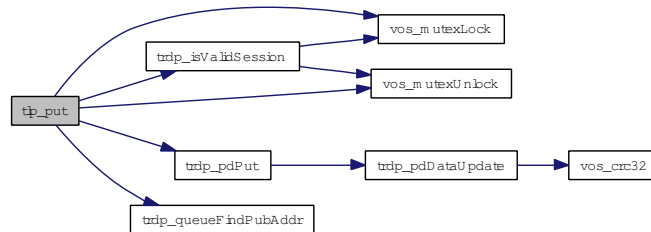
TRDP_PARAM_ERR parameter error

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:



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

Initiate sending PD messages (PULL).

Send a PD request message

Parameters:

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

← **subHandle** handle from related subscribe

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

- ← *topoCount* valid topocount, 0 for local consist
- ← *srcIpAddr* own IP address, 0 - srcIP will be set by the stack
- ← *destIpAddr* where to send the packet to
- ← *redId* 0 - Non-redundant, > 0 valid redundancy group
- ← *pktFlags* OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *pSendParam* optional pointer to send parameter, NULL - default parameters are used
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data
- ← *replyComId* comId of reply
- ← *replyIpAddr* IP for reply
- ← *subs* substitution (Ladder)
- ← *offsetAddr* offset (Ladder)

Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* parameter error
- TRDP_MEM_ERR* could not insert (out of memory)
- TRDP_NOINIT_ERR* handle invalid

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* OPTIONS: TRDP_FLAGS_MARSHALL, TRDP_FLAGS_CALLBACK
- ← *pSendParam* optional pointer to send parameter, NULL - default parameters are used
- ← *pData* pointer to packet data / dataset
- ← *dataSize* size of packet data
- ← *replyComId* comId of reply
- ← *replyIpAddr* IP for reply
- ← *subs* substitution (Ladder)
- ← *offsetAddr* offset (Ladder)

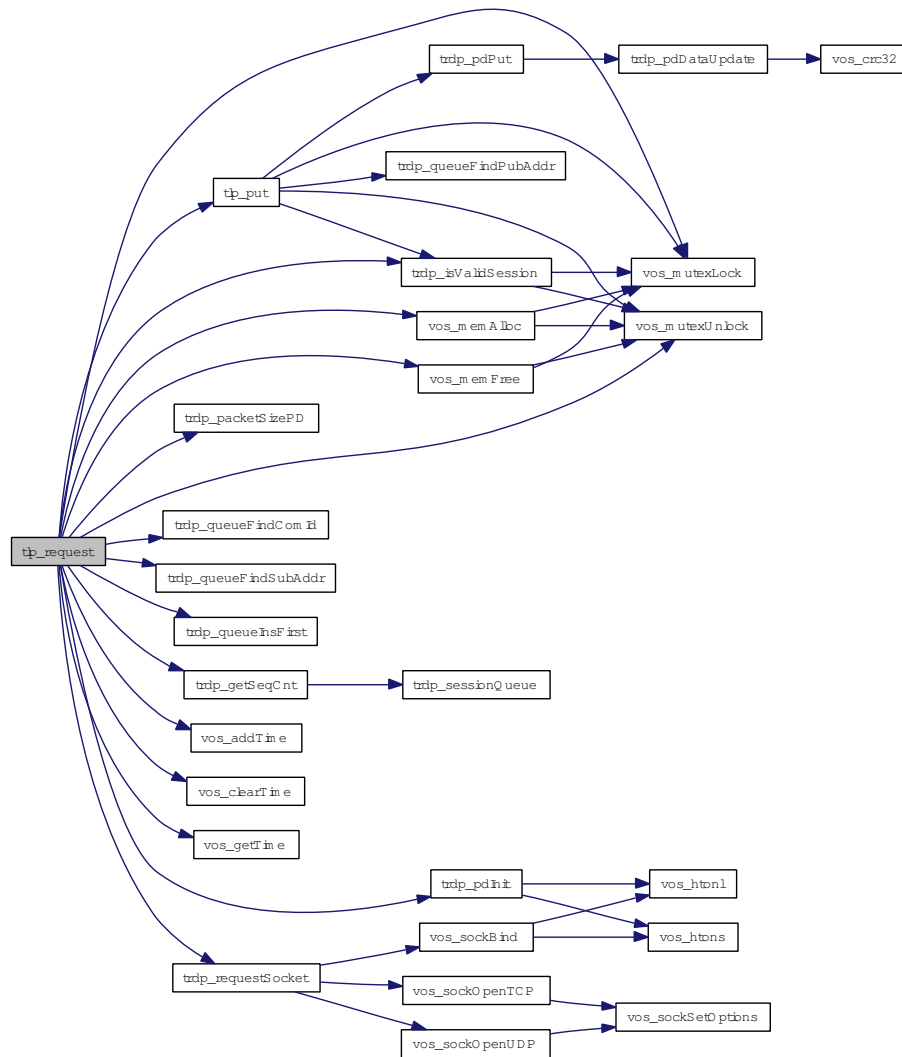
Return values:

- TRDP_NO_ERR* no error
- TRDP_PARAM_ERR* parameter error
- TRDP_MEM_ERR* could not insert (out of memory)

TRDP_NOINIT_ERR handle invalid

TRDP_NOSUB_ERR no matching subscription found

Here is the call graph for this function:



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

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

Parameters:

← **appHandle** the handle returned by tlc_init

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

← **leader** TRUE if we send

Return values:

TRDP_NO_ERR no error
TRDP_PARAM_ERR parameter error / redId not existing
TRDP_NOINIT_ERR handle invalid

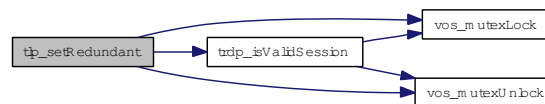
Parameters:

← **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:



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

Prepare for receiving PD messages.

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

Parameters:

← **appHandle** the handle returned by tlc_init
 → **pSubHandle** return a handle for these messages
 ← **pUserRef** user supplied value returned within the info structure
 ← **comId** comId of packet to receive
 ← **topoCount** valid topocount, 0 for local consist
 ← **srcIpAddr1** IP for source filtering, set 0 if not used
 ← **srcIpAddr2** Second source IP address for source filtering, set to zero if not used. Used e.g. for source filtering of redundant devices.
 ← **destIpAddr** IP address to join
 ← **timeout** timeout (≥ 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

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

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.

← *destIpAddr* IP address to join

← *timeout* timeout (>= 10ms) in usec

← *toBehavior* timeout behavior

← *maxDataSize* expected max. size of packet data

Return values:

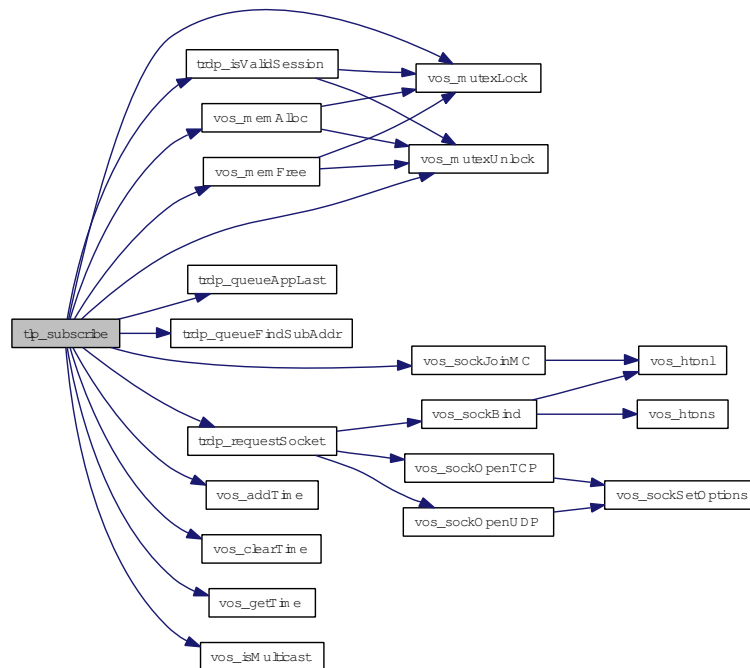
TRDP_NO_ERR no error

TRDP_PARAM_ERR parameter error

TRDP_MEM_ERR could not reserve memory (out of memory)

TRDP_NOINIT_ERR handle invalid

Here is the call graph for this function:



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

Stop sending PD messages.

Parameters:

- ← *appHandle* the handle returned by tlc_init
- ← *pubHandle* the handle returned by prepare

Return values:

- TRDP_NO_ERR** no error
- TRDP_PARAM_ERR** parameter error
- TRDP_NOPUB_ERR** not published
- TRDP_NOINIT_ERR** handle invalid

Parameters:

- ← *appHandle* the handle returned by tlc_openSession
- ← *pubHandle* the handle returned by prepare

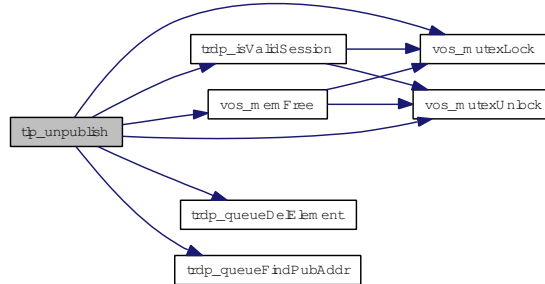
Return values:

- 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:



5.13.2.35 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

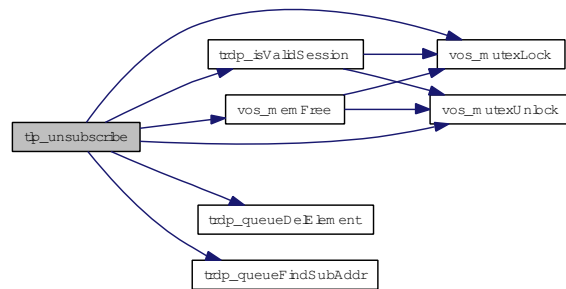
Parameters:

- ← *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_SUB_ERR** not subscribed
- TRDP_NOINIT_ERR** handle invalid

Here is the call graph for this function:

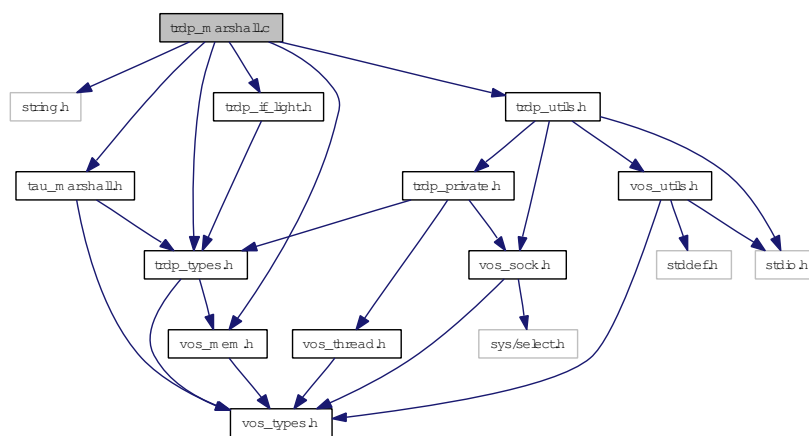


5.14 trdp_marshall.c File Reference

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 trdp_marshall.c:



Data Structures

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

5.14.1 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

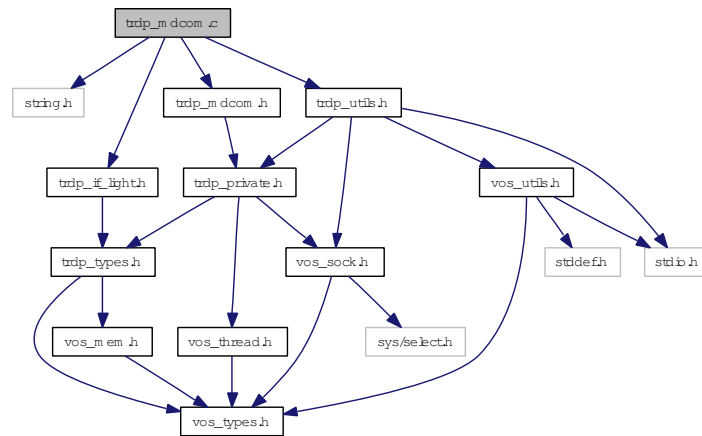
[trdp_marshall.c](#) 133 2012-11-07 17:38:51Z 97025

5.15 trdp_mdcom.c File Reference

Functions for MD communication.

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

Include dependency graph for trdp_mdcom.c:



Functions

- **TRDP_ERR_T** [trdp_sendMD](#) (int mdSock, const **MD_ELE_T** *pPacket)
Send MD packet.
- **TRDP_ERR_T** [trdp_rcvMD](#) (int mdSock, **MD_HEADER_T** **ppPacket, INT32 *pSize, UINT32 *pIPAddr)
Receive MD packet.
- **TRDP_ERR_T** [trdp_mdCheck](#) (**TRDP_SESSION_PT** appHandle, **MD_HEADER_T** *pPacket, INT32 packetSize)
Check for incoming md packet.
- void [trdp_mdUpdate](#) (**MD_ELE_T** *pPacket)
Update the header values.
- **TRDP_ERR_T** [trdp_mdSend](#) (INT32 pdSock, const **MD_ELE_T** *pPacket)
Send MD packet.
- **TRDP_ERR_T** [trdp_mdRecv](#) (**TRDP_SESSION_PT** appHandle, INT32 mdSock, **MD_ELE_T** *pPacket)
Receive MD packet.
- **TRDP_ERR_T** [trdp_mdReceive](#) (**TRDP_SESSION_PT** appHandle, INT32 sock)

*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.*

5.15.1 Detailed Description

Functions for MD communication.

Note:

Project: TCNOpen TRDP prototype stack

Author:

Simone Pachera, FARsystems

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_mdcom.c](#) 159 2012-11-20 16:51:12Z bloehr

5.15.2 Function Documentation

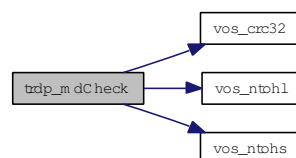
5.15.2.1 TRDP_ERR_T trdp_mdCheck (TRDP_SESSION_PT *appHandle*, MD_HEADER_T * *pPacket*, INT32 *packetSize*)

Check for incoming md packet.

Parameters:

- ← *appHandle* session pointer
- ← *pPacket* pointer to the packet to check
- ← *packetSize* size of the packet

Here is the call graph for this function:



5.15.2.2 TRDP_ERR_T trdp_mdReceive (TRDP_SESSION_PT *appHandle*, INT32 *sock*)

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.

Call user's callback if needed

Parameters:

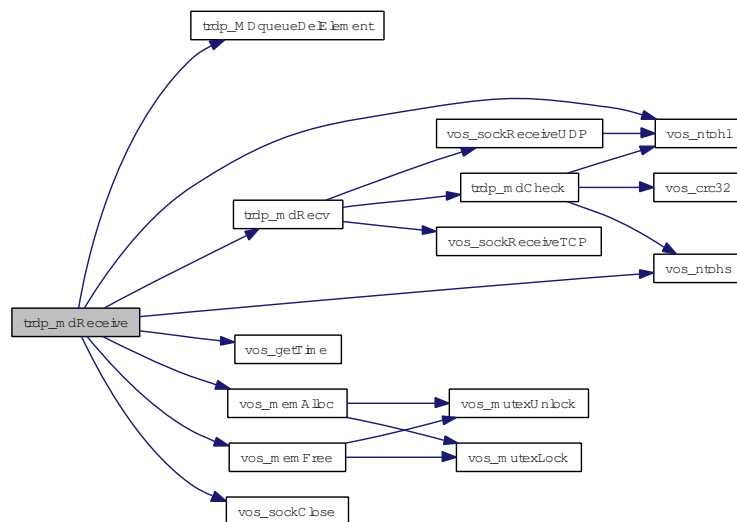
- ← *appHandle* session pointer
- ← *sock* the socket to read from

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:



5.15.2.3 TRDP_ERR_T trdp_mdRecv (TRDP_SESSION_PT *appHandle*, INT32 *mdSock*, MD_ELEM_T * *pPacket*)

Receive MD packet.

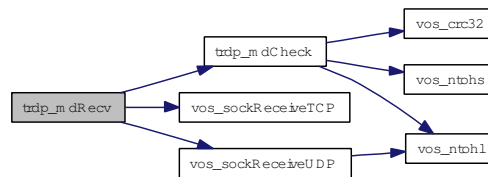
Parameters:

- ← *appHandle* session pointer
- ← *mdSock* socket descriptor
- ← *pPacket* pointer to received packet

Return values:

- != NULL error

Here is the call graph for this function:



5.15.2.4 TRDP_ERR_T trdp_mdSend (INT32 *pdSock*, const MD_ELE_T * *pPacket*)

Send MD packet.

Parameters:

- ← *pdSock* socket descriptor
- ← *pPacket* pointer to packet to be sent

Return values:

!= NULL error

Here is the call graph for this function:



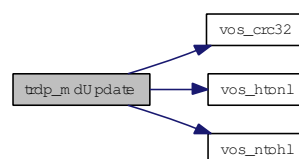
5.15.2.5 void trdp_mdUpdate (MD_ELE_T * *pPacket*)

Update the header values.

Parameters:

- ← *pPacket* pointer to the packet to update

Here is the call graph for this function:



5.15.2.6 TRDP_ERR_T trdp_rcvMD (int *mdSock*, MD_HEADER_T ** *ppPacket*, INT32 * *pSize*, UINT32 * *pIPAddr*)

Receive MD packet.

Parameters:

- ← *mdSock* socket descriptor
- *ppPacket* pointer to pointer to received packet
- *pSize* pointer to size of received packet
- *pIPAddr* pointer to source IP address of packet

Return values:

- TRDP_NO_ERR* no error
- TRDP_UNKNOWN_ERR* error

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

Send MD packet.

Parameters:

- ← *mdSock* socket descriptor
- ← *pPacket* pointer to packet to be sent

Return values:

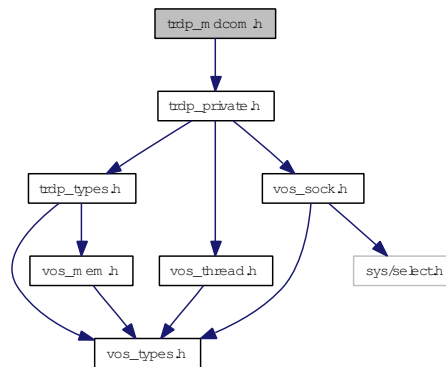
- TRDP_NO_ERR* no error
- TRDP_UNKNOWN_ERR* error

5.16 trdp_mdcom.h File Reference

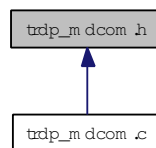
Functions for MD communication.

```
#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_mdSend (INT32 pdSock, const **MD_ELE_T** *pPacket)
Send MD packet.
- void trdp_mdUpdate (**MD_ELE_T** *pPacket)
Update the header values.
- **TRDP_ERR_T** trdp_mdReceive (**TRDP_SESSION_PT** appHandle, INT32 sock)
*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.*

5.16.1 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, 2012.

Id

[trdp_mdcom.h](#) 120 2012-11-06 17:32:14Z 97025

5.16.2 Function Documentation**5.16.2.1 TRDP_ERR_T trdp_mdReceive (TRDP_SESSION_PT *appHandle*, INT32 *sock*)**

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.

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

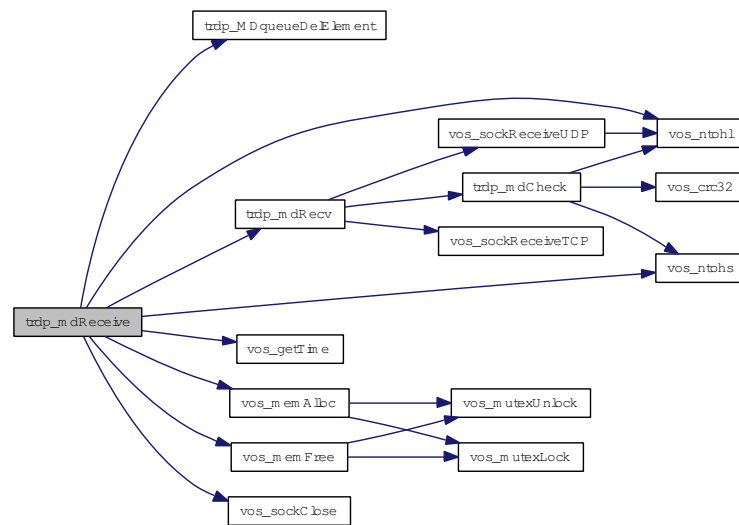
TRDP_CRC_ERR header checksum

TRDP_TOPOCOUNT_ERR invalid topocount

—

—

Here is the call graph for this function:



5.16.2.2 TRDP_ERR_T trdp_mdSend (INT32 *pdSock*, const MD_ELE_T * *pPacket*)

Send MD packet.

Parameters:

- ← *pdSock* socket descriptor
- ← *pPacket* pointer to packet to be sent

Return values:

!= NULL error

Here is the call graph for this function:



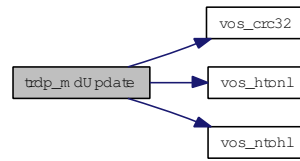
5.16.2.3 void trdp_mdUpdate (MD_ELE_T * *pPacket*)

Update the header values.

Parameters:

- ← *pPacket* pointer to the packet to update

Here is the call graph for this function:

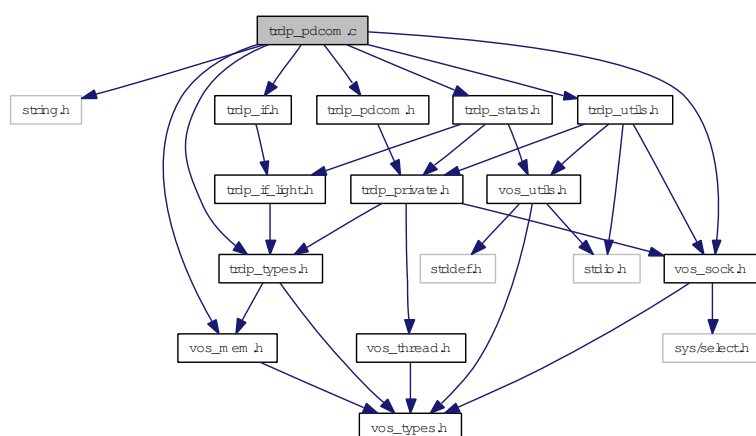


5.17 trdp_pdcom.c File Reference

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, UINT16 subs, UINT16 offsetAddress, UINT32 replyComId, 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 dataSize)
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_pdUpdate](#) ([PD_ELE_T](#) *pPacket)
Update the header values.
- [TRDP_ERR_T trdp_pdCheck](#) ([PD_HEADER_T](#) *pPacket, INT32 packetSize)
Check if the PD header values and the CRCs are sane.
- [TRDP_ERR_T trdp_pdSend](#) (INT32 pdSock, [PD_ELE_T](#) *pPacket)
Send one PD packet.
- [TRDP_ERR_T trdp_pdDistribute](#) ([PD_ELE_T](#) *pSndQueue)
Distribute send time of PD packets over time.

5.17.1 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](#) 159 2012-11-20 16:51:12Z bloehr

5.17.2 Function Documentation

5.17.2.1 [TRDP_ERR_T trdp_pdCheck](#) ([PD_HEADER_T](#) *pPacket, INT32 packetSize)

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

Parameters:

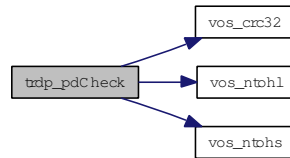
- ← *pPacket* pointer to the packet to check
- ← *packetSize* max size to check

Return values:

TRDP_NO_ERR

TRDP_CRC_ERR

Here is the call graph for this function:

**5.17.2.2 void trdp_pdDataUpdate (PD_ELE_T * pPacket)**

Add padding and update data CRC.

Here is the call graph for this function:

**5.17.2.3 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.

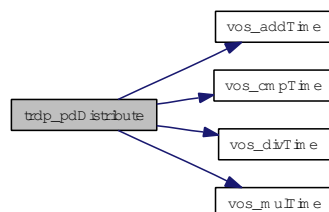
Parameters:

← *pSndQueue* pointer to send queue

Return values:

TRDP_NO_ERR

Here is the call graph for this function:



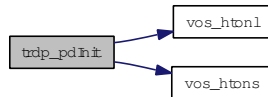
5.17.2.4 void trdp_pdInit (PD_ELE_T * *pPacket*, TRDP_MSG_T *type*, UINT32 *topoCount*, UINT16 *subs*, UINT16 *offsetAddress*, UINT32 *replyComId*, UINT32 *replyIpAddress*)

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
- ← *subs* subsAndReserve
- ← *offsetAddress* ladder offset
- ← *replyComId* Pull request comId
- ← *replyIpAddress* Pull request Ip

Here is the call graph for this function:



5.17.2.5 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

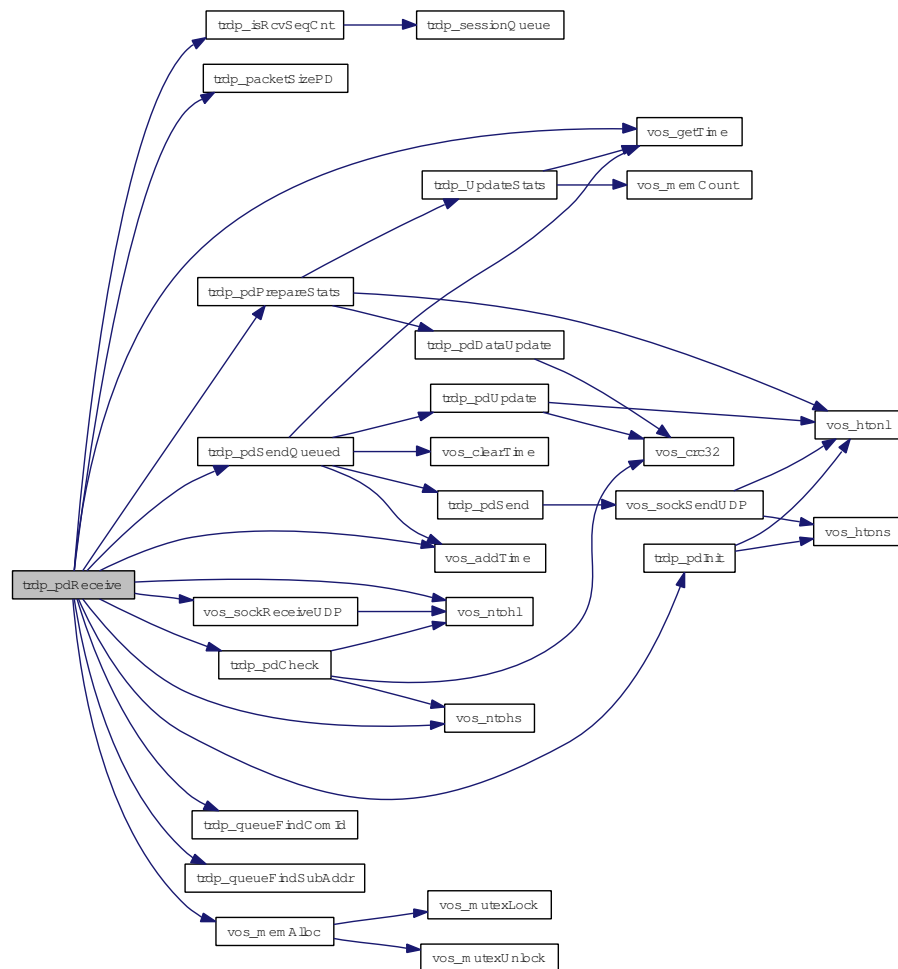
Parameters:

- ← *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:



5.17.2.6 TRDP_ERR_T trdp_pdSend (INT32 *pdSock*, PD_ELE_T * *pPacket*)

Send one PD packet.

Parameters:

← *pdSock* socket descriptor

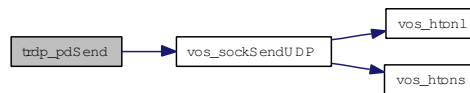
← *pPacket* pointer to packet to be sent

Return values:

TRDP_NO_ERR

TRDP_IO_ERR

Here is the call graph for this function:



5.17.2.7 TRDP_ERR_T trdp_pdSendQueued (TRDP_SESSION_PT *appHandle*)

Send all due PD messages.

Parameters:

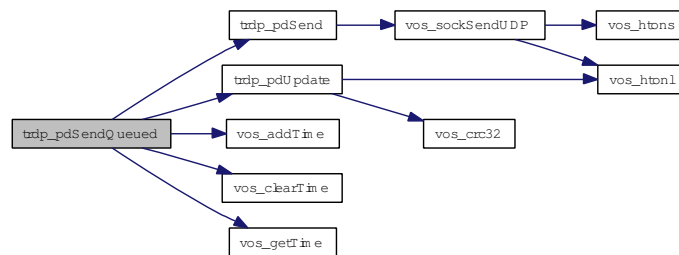
← *appHandle* session pointer

Return values:

TRDP_NO_ERR no error

TRDP_IO_ERR socket I/O error

Here is the call graph for this function:



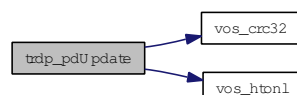
5.17.2.8 void trdp_pdUpdate (PD_ELE_T * *pPacket*)

Update the header values.

Parameters:

← *pPacket* pointer to the packet to update

Here is the call graph for this function:

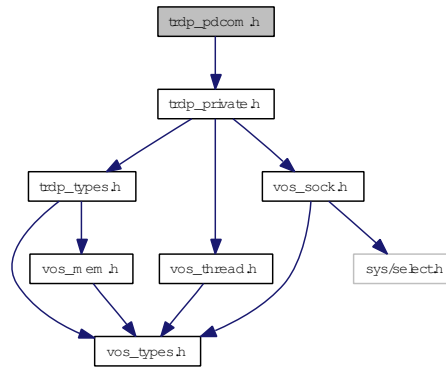


5.18 trdp_pdcom.h File Reference

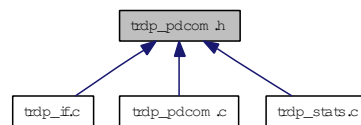
Functions for PD communication.

```
#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, UINT16 subs, UINT16 offsetAddress, 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, INT32 packetSize)
Check if the PD header values and the CRCs are sane.
- [TRDP_ERR_T](#) [trdp_pdSend](#) (INT32 sock, [PD_ELE_T](#) *)
Send one PD packet.

- [TRDP_ERR_T trdp_pdGet](#) ([PD_ELE_T](#) *pPacket, [TRDP_UNMARSHALL_T](#) unmarshall, void *refCon, const [UINT8](#) *pData, [UINT32](#) dataSize)
Copy data Set the header infos.
- [TRDP_ERR_T trdp_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.*
- [TRDP_ERR_T trdp_pdDistribute](#) ([PD_ELE_T](#) *pSndQueue)
Distribute send time of PD packets over time.

5.18.1 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](#) 53 2012-10-17 17:40:43Z 97025

5.18.2 Function Documentation

5.18.2.1 [TRDP_ERR_T trdp_pdCheck](#) ([PD_HEADER_T](#) *pPacket, [INT32](#) packetSize)

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

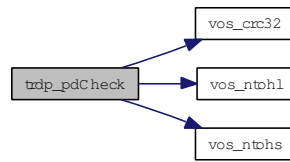
Parameters:

- ← *pPacket* pointer to the packet to check
- ← *packetSize* max size to check

Return values:

TRDP_NO_ERR
TRDP_CRC_ERR

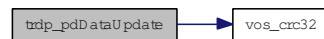
Here is the call graph for this function:



5.18.2.2 void trdp_pdDataUpdate (PD_ELE_T *pPacket)

Add padding and update data CRC.

Here is the call graph for this function:



5.18.2.3 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.

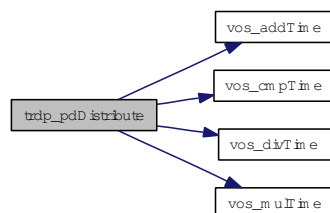
Parameters:

← *pSndQueue* pointer to send queue

Return values:

TRDP_NO_ERR

Here is the call graph for this function:



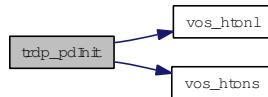
5.18.2.4 void trdp_pdInit (PD_ELE_T * *pPacket*, TRDP_MSG_T *type*, UINT32 *topoCount*, UINT16 *subs*, UINT16 *offsetAddress*, UINT32 *replyComId*, UINT32 *replyIpAddress*)

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
- ← *subs* subsAndReserve
- ← *offsetAddress* ladder offset
- ← *replyComId* Pull request comId
- ← *replyIpAddress* Pull request Ip

Here is the call graph for this function:



5.18.2.5 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

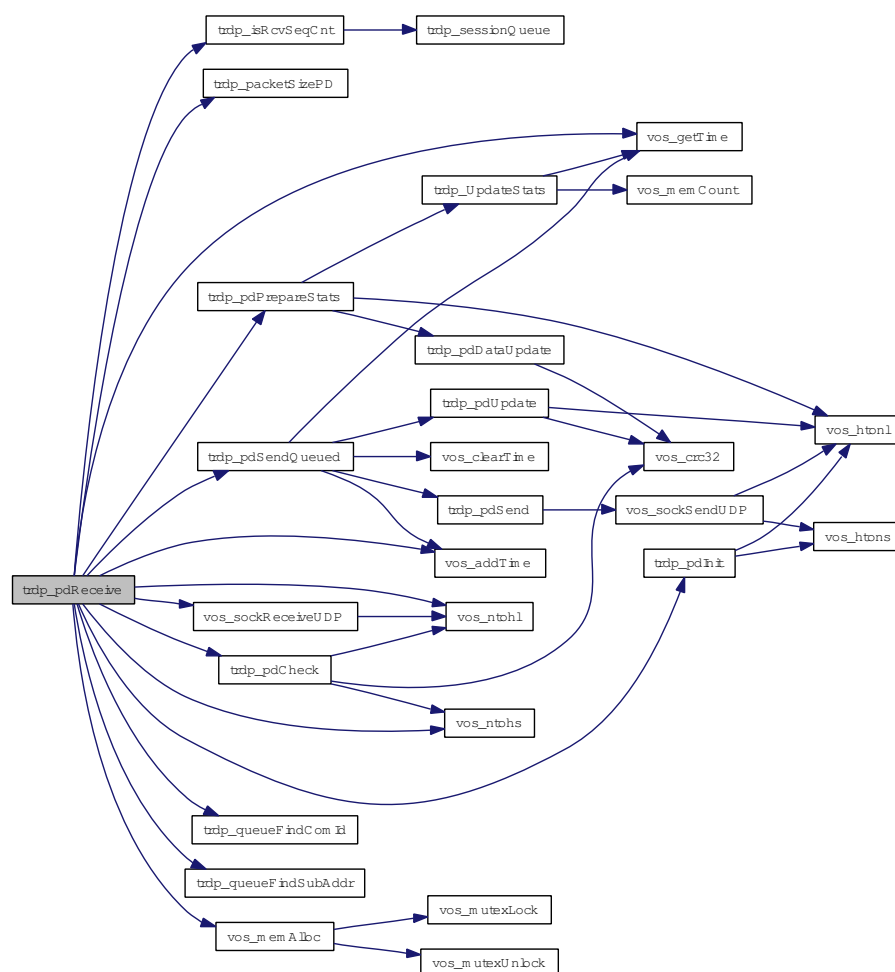
Parameters:

- ← *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:



5.18.2.6 TRDP_ERR_T trdp_pdSend (INT32 *pdSock*, PD_ELE_T * *pPacket*)

Send one PD packet.

Parameters:

← *pdSock* socket descriptor

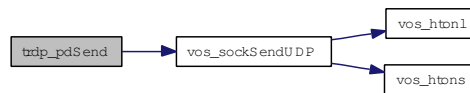
← *pPacket* pointer to packet to be sent

Return values:

TRDP_NO_ERR

TRDP_IO_ERR

Here is the call graph for this function:



5.18.2.7 TRDP_ERR_T trdp_pdSendQueued (TRDP_SESSION_PT *appHandle*)

Send all due PD messages.

Parameters:

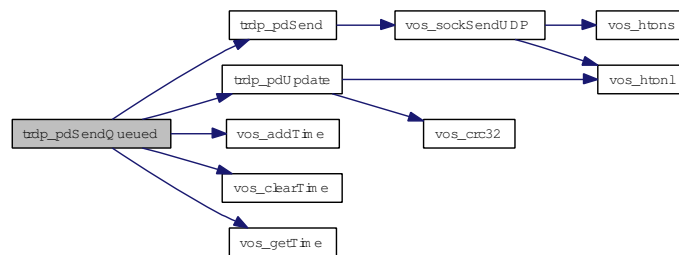
← *appHandle* session pointer

Return values:

TRDP_NO_ERR no error

TRDP_IO_ERR socket I/O error

Here is the call graph for this function:



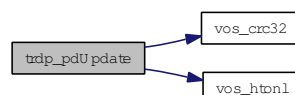
5.18.2.8 void trdp_pdUpdate (PD_ELE_T * *pPacket*)

Update the header values.

Parameters:

← *pPacket* pointer to the packet to update

Here is the call graph for this function:



5.19 trdp_private.h File Reference

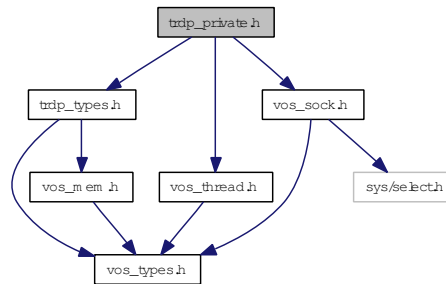
Typedefs for TRDP communication.

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

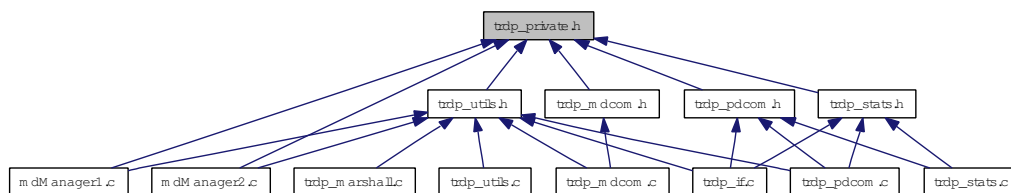
```
#include "vos_thread.h"
```

```
#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_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 [GNU_PACKED](#)
TRDP process data header - network order and alignment.
- struct [PD_ELE](#)
Queue element for PD packets to send or receive.

- struct [MD_ELE](#)
Queue element for MD packets to send or receive or acknowledge.
- struct [TRDP_SESSION](#)
Session/application variables store.

Defines

- #define [IP_PD_UDP_PORT](#) 20548
process data UDP port
- #define [IP_MD_UDP_PORT](#) 20550
message data UDP port
- #define [IP_MD_TCP_PORT](#) 20550
message data TCP port
- #define [IP_PD_PROTO_VER](#) 0x0100
Protocol version.
- #define [TIMER_GRANULARITY](#) 10000
granularity in us
- #define [MD_DEFAULT_REPLY_TIMEOUT](#) 10000000
default reply time out 10s
- #define [MD_DEFAULT_CONFIRM_TIMEOUT](#) 10000000
default reply time out 10s
- #define [MIN_PD_HEADER_SIZE](#) sizeof(PD_HEADER_T)
PD header size with FCS.
- #define [ACK_TIME_OUT_VAL_DEF](#) 500
Default value in milliseconds for waiting on acknowledge message.

Typedefs

- typedef struct [TRDP_HANDLE](#) [TRDP_ADDRESSES_T](#)
Hidden handle definition, used as unique addressing item.
- typedef struct [TRDP_SOCKETS](#) [TRDP_SOCKETS_T](#)
Socket item.
- typedef struct [PD_ELE](#) [PD_ELE_T](#)
Queue element for PD packets to send or receive.
- typedef struct [MD_ELE](#) [MD_ELE_T](#)

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

- typedef struct [TRDP_SESSION](#) [TRDP_SESSION_T](#)
Session/application variables store.

Enumerations

- enum [TRDP_MD_ELE_ST_T](#) {
[TRDP_MD_ELE_ST_NONE](#) = 0,
[TRDP_MD_ELE_ST_TX_NOTIFY_ARM](#) = 1,
[TRDP_MD_ELE_ST_TX_REQUEST_ARM](#) = 2,
[TRDP_MD_ELE_ST_TX_REPLY_ARM](#) = 3,
[TRDP_MD_ELE_ST_TX_REPLYQUERY_ARM](#) = 4,
[TRDP_MD_ELE_ST_TX_CONFIRM_ARM](#) = 5,
[TRDP_MD_ELE_ST_TX_ERROR_ARM](#) = 6,
[TRDP_MD_ELE_ST_TX_REQUEST_W4Y](#) = 7,
[TRDP_MD_ELE_ST_TX_REPLYQUERY_W4C](#) = 8,
[TRDP_MD_ELE_ST_RX_ARM](#) = 9,
[TRDP_MD_ELE_ST_RX_REQ_W4AP_REPLY](#) = 10,
[TRDP_MD_ELE_ST_RX_REPLY_W4AP_CONF](#) = 11 }
Internal MD state.

- enum [TRDP_PRIV_FLAGS_T](#) { ,
[TRDP_TIMED_OUT](#) = 0x2,
[TRDP_REQ_2B_SENT](#) = 0x4,
[TRDP_PULL_SUB](#) = 0x8 }
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.

5.19.1 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](#) 145 2012-11-16 18:45:07Z bloehr

5.19.2 Enumeration Type Documentation**5.19.2.1 enum TRDP_MD_ELE_ST_T**

Internal MD state.

Enumerator:

TRDP_MD_ELE_ST_NONE neutral value
TRDP_MD_ELE_ST_TX_NOTIFY_ARM ready to send notify MD
TRDP_MD_ELE_ST_TX_REQUEST_ARM ready to send request MD
TRDP_MD_ELE_ST_TX_REPLY_ARM ready to send reply MD
TRDP_MD_ELE_ST_TX_REPLYQUERY_ARM ready to send reply with confirm request MD
TRDP_MD_ELE_ST_TX_CONFIRM_ARM ready to send confirm MD
TRDP_MD_ELE_ST_TX_ERROR_ARM ready to send error MD
TRDP_MD_ELE_ST_TX_REQUEST_W4Y request sent, wait for reply
TRDP_MD_ELE_ST_TX_REPLYQUERY_W4C reply send, with confirm request MD
TRDP_MD_ELE_ST_RX_ARM armed listener
TRDP_MD_ELE_ST_RX_REQ_W4AP_REPLY request received, wait for application reply send
TRDP_MD_ELE_ST_RX_REPLY_W4AP_CONF reply conf.
 rq. rx, wait for application conf send

5.19.2.2 enum TRDP_PRIV_FLAGS_T

Internal flags for packets.

Enumerator:

TRDP_TIMED_OUT if set, informed the user
TRDP_REQ_2B_SENT if set, the request needs to be sent
TRDP_PULL_SUB if set, its a PULL subscription

5.19.2.3 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.

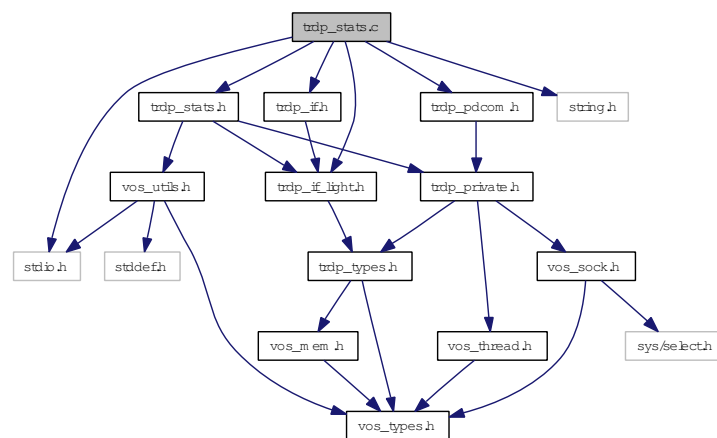
TRDP SOCK_MD_TCP Socket is used for TCP message data.

5.20 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 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_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.

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

Fill the statistics packet.

5.20.1 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](#) 138 2012-11-14 11:34:11Z 97031

5.20.2 Function Documentation

5.20.2.1 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.

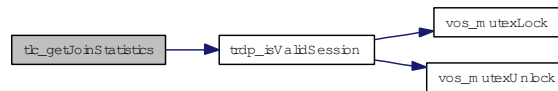
Parameters:

- ← **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 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:



5.20.2.2 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.

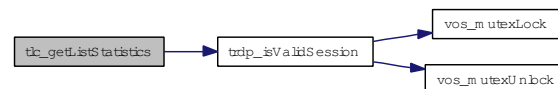
Parameters:

← **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:



5.20.2.3 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:

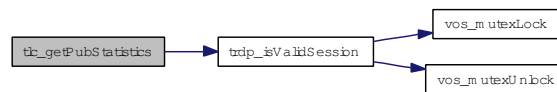
← **appHandle** the handle returned by tlc_openSession

- ↔ *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:



5.20.2.4 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.

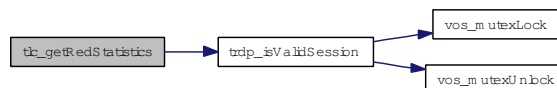
Parameters:

- ← *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:



5.20.2.5 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.

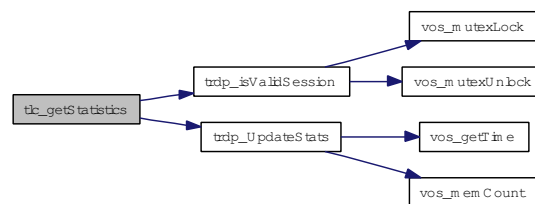
Parameters:

- ← *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:



5.20.2.6 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.

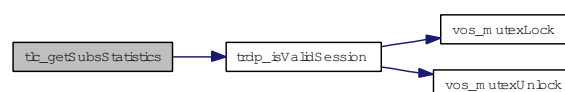
Parameters:

- ← *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:



5.20.2.7 EXT_DECL TRDP_ERR_T tlc_resetStatistics (TRDP_APP_SESSION_T *appHandle*)

Reset statistics.

Parameters:

← *appHandle* the handle returned by tlc_openSession

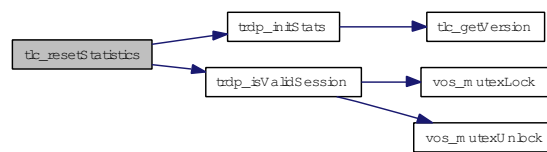
Return values:

TRDP_NO_ERR no error

TRDP_NOINIT_ERR handle invalid

TRDP_PARAM_ERR parameter error

Here is the call graph for this function:



5.20.2.8 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

Here is the call graph for this function:



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

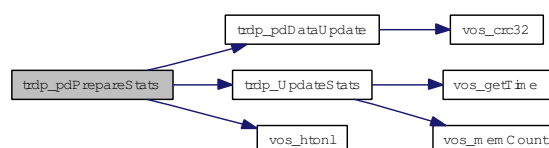
Fill the statistics packet.

Parameters:

← *appHandle* the handle returned by tlc_openSession

↔ *pPacket* pointer to the packet to fill

Here is the call graph for this function:



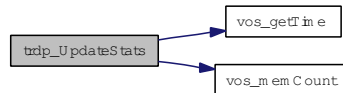
5.20.2.10 void trdp_UpdateStats (TRDP_APP_SESSION_T *appHandle*)

Update the statistics.

Parameters:

← *appHandle* the handle returned by tlc_openSession

Here is the call graph for this function:



5.21 trdp_stats.h File Reference

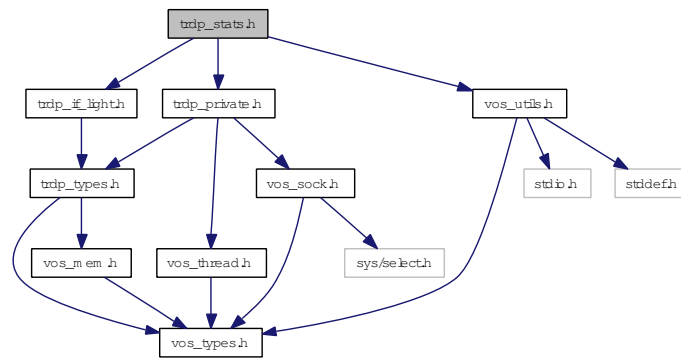
Statistics for TRDP communication.

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

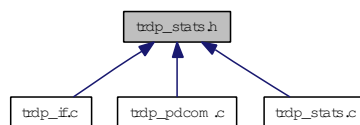
```
#include "trdp_private.h"
```

```
#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.

5.21.1 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](#) 53 2012-10-17 17:40:43Z 97025

5.21.2 Function Documentation**5.21.2.1 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

Here is the call graph for this function:

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

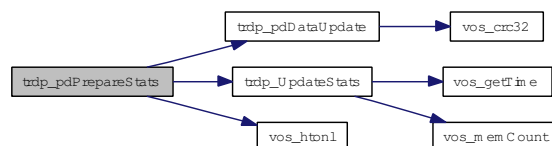
Fill the statistics packet.

Parameters:

← *appHandle* the handle returned by tlc_openSession

↔ *pPacket* pointer to the packet to fill

Here is the call graph for this function:



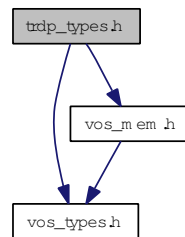
5.22 trdp_types.h File Reference

Typedefs for TRDP communication.

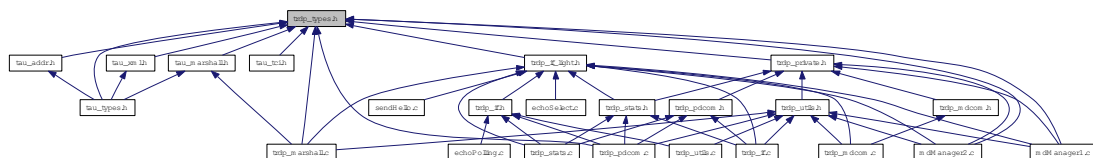
```
#include "vos_types.h"
```

```
#include "vos_mem.h"
```

Include dependency graph for trdp_types.h:



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



Data Structures

- struct [TRDP_PD_INFO_T](#)
Process data info from received telegram; allows the application to generate responses.
- struct [TRDP_MD_INFO_T](#)
Message data info from received telegram; allows the application to generate responses.
- struct [TRDP_SEND_PARAM_T](#)
Quality/type of service and time to live.
- struct [TRDP_DATASET_ELEMENT_T](#)
Dataset element definition.
- struct [TRDP_DATASET_T](#)
Dataset definition.
- struct [TRDP_COMID_DSID_MAP_T](#)
Dataset 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](#)
Structure describing memory (and its pre-fragmentation).
- struct [TRDP_PROCESS_CONFIG_T](#)
Types to read out the XML configuration.

Defines

- #define [TRDP_MAX_LABEL_LEN](#) 16
Maximum values.
- #define [TRDP_MAX_URI_USER_LEN](#) (2 * TRDP_MAX_LABEL_LEN)
URI user part incl.
- #define [TRDP_MAX_URI_HOST_LEN](#) (4 * TRDP_MAX_LABEL_LEN)
URI host part length incl.
- #define [TRDP_MAX_URI_LEN](#) ((6 * TRDP_MAX_LABEL_LEN) + 8)

URI length incl.

- #define [TRDP_MAX_FILE_NAME_LEN](#) 128
path and file name length incl.
- #define [TRDP_VAR_SIZE](#) 0
Variable size dataset.
- #define [USE_HEAP](#) 0
If this is set, we can allocate dynamically memory.
- #define [TRDP_COMID_ECHO](#) 10
TRDP reserved COMID's in the range 1 .
- #define [TRDP_STATISTICS_REQUEST_DSID](#) 31
TRDP reserved data set id's in the range 1 .

Typedefs

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

- typedef void(* [TRDP_MD_CALLBACK_T](#))(void *pRefCon, const [TRDP_MD_INFO_T](#) *pMsg, UINT8 *pData, UINT32 dataSize)
Callback for receiving indications, timeouts, releases, responses.
- typedef [VOS_MEM_BLK_T](#) [TRDP_MEM_BLK_T](#)
Enumeration type for memory pre-fragmentation, reuse of VOS definition.

Enumerations

- enum [TRDP_ERR_T](#) {
[TRDP_NO_ERR](#) = 0,
[TRDP_PARAM_ERR](#) = -1,
[TRDP_INIT_ERR](#) = -2,
[TRDP_NOINIT_ERR](#) = -3,
[TRDP_TIMEOUT_ERR](#) = -4,
[TRDP_NODATA_ERR](#) = -5,
[TRDP_SOCKET_ERR](#) = -6,
[TRDP_IO_ERR](#) = -7,
[TRDP_MEM_ERR](#) = -8,
[TRDP_SEMA_ERR](#) = -9,
[TRDP_QUEUE_ERR](#) = -10,
[TRDP_QUEUE_FULL_ERR](#) = -11,
[TRDP_MUTEX_ERR](#) = -12,
[TRDP_NOSESSION_ERR](#) = -13,
[TRDP_SESSION_ABORT_ERR](#) = -14,
[TRDP_NOSUB_ERR](#) = -15,
[TRDP_NOPUB_ERR](#) = -16,
[TRDP_NOLIST_ERR](#) = -17,
[TRDP_CRC_ERR](#) = -18 ,
[TRDP_TOPO_ERR](#) = -20,
[TRDP_COMID_ERR](#) = -21,
[TRDP_STATE_ERR](#) = -22,
[TRDP_APPTIMEOUT_ERR](#) = -23,
[TRDP_UNKNOWN_ERR](#) = -99 }
Return codes for all API functions.
- enum [TRDP_MSG_T](#) {
[TRDP_MSG_PD](#) = 0x5064,
[TRDP_MSG_PR](#) = 0x5072,
[TRDP_MSG_PE](#) = 0x5065,
[TRDP_MSG_MN](#) = 0x4D6E,
[TRDP_MSG_MR](#) = 0x4D72,

```
TRDP_MSG_MP = 0x4D70,  
TRDP_MSG_MQ = 0x4D71,  
TRDP_MSG_MC = 0x4D63,  
TRDP_MSG_ME = 0x4D65 }
```

TRDP data transfer type definitions.

- enum `TRDP_REPLY_STATUS_T`

Reply status messages.

- enum `TRDP_FLAGS_T` { ,
 `TRDP_FLAGS_REDUNDANT` = 0x1,
 `TRDP_FLAGS_MARSHALL` = 0x2,
 `TRDP_FLAGS_CALLBACK` = 0x4,
 `TRDP_FLAGS_TCP` = 0x8 }

Various flags for PD and MD packets.

- enum `TRDP_RED_STATE_T` {
 `TRDP_RED_FOLLOWER` = 0,
 `TRDP_RED_LEADER` = 1 }

Redundancy states.

- enum `TRDP_TO_BEHAVIOR_T` {
 `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.

5.22.1 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` 159 2012-11-20 16:51:12Z bloehr

5.22.2 Define Documentation

5.22.2.1 #define TRDP_COMID_ECHO 10

TRDP reserved COMID's in the range 1 .

.. 1000

5.22.2.2 #define TRDP_MAX_FILE_NAME_LEN 128

path and file name length incl.

terminating '0'

5.22.2.3 #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: $7 + (6 * 15) + 5 * (\text{sizeof}(\text{separator})) + 1$ (terminating 0) to facilitate alignment the size
will be increased by 1 byte label length incl. terminating '0'

5.22.2.4 #define TRDP_MAX_URI_HOST_LEN (4 * TRDP_MAX_LABEL_LEN)

URI host part length incl.

terminating '0'

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

URI length incl.

terminating '0' and 1 padding byte

5.22.2.6 #define TRDP_MAX_URI_USER_LEN (2 * TRDP_MAX_LABEL_LEN)

URI user part incl.

terminating '0'

5.22.2.7 #define TRDP_STATISTICS_REQUEST_DSID 31

TRDP reserved data set id's in the range 1 .

.. 1000

5.22.3 Typedef Documentation**5.22.3.1 typedef UINT32 TRDP_IP_ADDR_T**

TRDP general type definitions.

5.22.3.2 typedef TRDP_ERR_T(* TRDP_MARSHALL_T)(void *pRefCon, UINT32 comId, UINT8 *pSrc, UINT8 *pDst, UINT32 *pDstSize)

Function type for marshalling .

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

Parameters:

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

Return values:

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

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

Callback for receiving indications, timeouts, releases, responses.

Parameters:

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

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

Callback for receiving indications, timeouts, releases, responses.

Parameters:

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

5.22.3.5 `typedef VOS_PRINT_DBG_T TRDP_PRINT_DBG_T`

TRDP configuration type definitions.

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

5.22.3.6 `typedef VOS_TIME_T TRDP_TIME_T`

Timer value compatible with timeval / select.

Relative or absolute date, depending on usage

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

Function type for unmarshalling.

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

Parameters:

- ← **pRefCon* pointer to user context
- ← *comId* ComId to identify the structure out of a configuration
- ← **pSrc* pointer to received original message
- ← **pDst* pointer to a buffer for the treated message

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

Return values:

TRDP_NO_ERR no error

TRDP_MEM_ERR provide buffer to small

TRDP_COMID_ERR comid not existing

5.22.4 Enumeration Type Documentation

5.22.4.1 enum TRDP_DATA_TYPE_T

TRDP dataset description definitions.

Dataset element definition

Enumerator:

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

TRDP_CHAR8 char, can be used also as UTF8

TRDP_UTF16 Unicode UTF-16 character.

TRDP_INT8 Signed integer, 8 bit.

TRDP_INT16 Signed integer, 16 bit.

TRDP_INT32 Signed integer, 32 bit.

TRDP_INT64 Signed integer, 64 bit.

TRDP_UINT8 Unsigned integer, 8 bit.

TRDP_UINT16 Unsigned integer, 16 bit.

TRDP_UINT32 Unsigned integer, 32 bit.

TRDP_UINT64 Unsigned integer, 64 bit.

TRDP_REAL32 Floating point real, 32 bit.

TRDP_REAL64 Floating point real, 64 bit.

TRDP_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.

5.22.4.2 enum TRDP_ERR_T

Return codes for all API functions.

Enumerator:

TRDP_NO_ERR No error.

TRDP_PARAM_ERR Parameter missing or out of range.

TRDP_INIT_ERR Call without valid initialization.

TRDP_NOINIT_ERR Call with invalid handle.

TRDP_TIMEOUT_ERR 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_NOSESSION_ERR No such session.
TRDP_SESSION_ABORT_ERR Session aborted.
TRDP_NOSUB_ERR No subscriber.
TRDP_NOPUB_ERR No publisher.
TRDP_NOLIST_ERR No listener.
TRDP_CRC_ERR Wrong CRC.
TRDP_TOPO_ERR Invalid topo count.
TRDP_COMID_ERR Unknown ComId.
TRDP_STATE_ERR Call in wrong state.
TRDP_APPTIMEOUT_ERR Application Timeout.
TRDP_UNKNOWN_ERR Unspecified error.

5.22.4.3 enum TRDP_FLAGS_T

Various flags for PD and MD packets.

Enumerator:

TRDP_FLAGS_REDUNDANT Redundant.
TRDP_FLAGS_MARSHALL Optional marshalling/unmarshalling in TRDP stack.
TRDP_FLAGS_CALLBACK Use of callback function.
TRDP_FLAGS_TCP Use TCP for message data.

5.22.4.4 enum TRDP_MSG_T

TRDP data transfer type definitions.

Message Types

Enumerator:

TRDP_MSG_PD 'Pd' PD Data (Reply)
TRDP_MSG_PR 'Pr' PD Request
TRDP_MSG_PE 'Pe' PD Error
TRDP_MSG_MN 'Mn' MD Notification (Request without reply)
TRDP_MSG_MR 'Mr' MD Request with reply
TRDP_MSG_MP 'Mp' MD Reply without confirmation
TRDP_MSG_MQ 'Mq' MD Reply with confirmation
TRDP_MSG_MC 'Mc' MD Confirm
TRDP_MSG_ME 'Me' MD Error

5.22.4.5 enum TRDP_OPTION_T

Various flags/general TRDP options for library initialization.

Enumerator:

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

TRDP_OPTION_TRAFFIC_SHAPING Use traffic shaping - distribute packet sending.

5.22.4.6 enum TRDP_RED_STATE_T

Redundancy states.

Enumerator:

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

TRDP_RED_LEADER Redundancy leader - redundant PD will be sent out.

5.22.4.7 enum TRDP_TO_BEHAVIOR_T

How invalid PD shall be handled.

Enumerator:

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

TRDP_TO_KEEP_LAST_VALUE If not set, last received values will be returned.

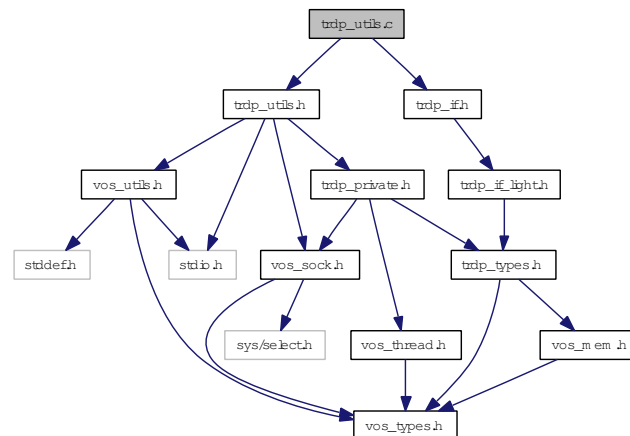
5.23 trdp_utils.c File Reference

Helper functions for TRDP communication.

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

```
#include "trdp_if.h"
```

Include dependency graph for trdp_utils.c:



Functions

- `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.
- `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_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[], const [TRDP_SEND_PARAM_T](#) *params, [TRDP_IP_ADDR_T](#) srcIP, [TRDP SOCK_TYPE_T](#) usage, [TRDP_OPTION_T](#) options, [BOOL](#) rcvOnly, [INT32](#) *pIndex)
Handle the socket pool: Request a socket from our socket pool.
- [TRDP_ERR_T](#) [trdp_releaseSocket](#) ([TRDP_SOCKETS_T](#) iface[], [INT32](#) index)
Handle the socket pool: Release a socket from our socket pool.
- [UINT32](#) [trdp_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.

5.23.1 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, 2012.

Id

[trdp_utils.c](#) 139 2012-11-14 15:15:56Z goiarbide

5.23.2 Function Documentation

5.23.2.1 int am_big_endian ()

Determine if we are Big or Little endian.

Return values:

!= 0 we are big endian

0 we are little endian

5.23.2.2 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:



5.23.2.3 void trdp_initSockets (TRDP_SOCKETS_T *iface*[])

Handle the socket pool: Initialize it.

Parameters:

← *iface* pointer to the socket pool

5.23.2.4 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:



5.23.2.5 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

5.23.2.6 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

5.23.2.7 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

5.23.2.8 `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

5.23.2.9 `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

5.23.2.10 `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

5.23.2.11 `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

5.23.2.12 `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

5.23.2.13 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

5.23.2.14 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

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

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

Parameters:

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

Return values:

- TRDP_NO_ERR*
- TRDP_PARAM_ERR*

Here is the call graph for this function:



5.23.2.16 `TRDP_ERR_T trdp_requestSocket (TRDP_SOCKETS_T iface[],
const TRDP_SEND_PARAM_T * params, TRDP_IP_ADDR_T srcIP,
TRDP SOCK_TYPE_T usage, TRDP_OPTION_T options, BOOL rcvOnly, INT32 *
pIndex)`

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

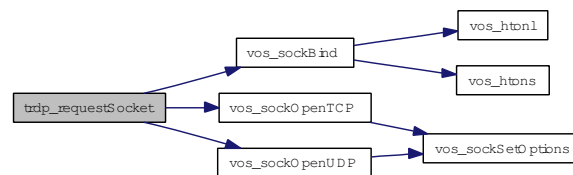
Parameters:

- ↔ *iface* socket pool
- ← *params* parameters to use
- ← *srcIP* IP to bind to (0 = any address)
- ← *usage* type and port to bind to
- ← *options* blocking/nonblocking
- ← *rcvOnly* only used for receiving
- *pIndex* returned index of socket pool

Return values:

TRDP_NO_ERR
TRDP_PARAM_ERR

Here is the call graph for this function:

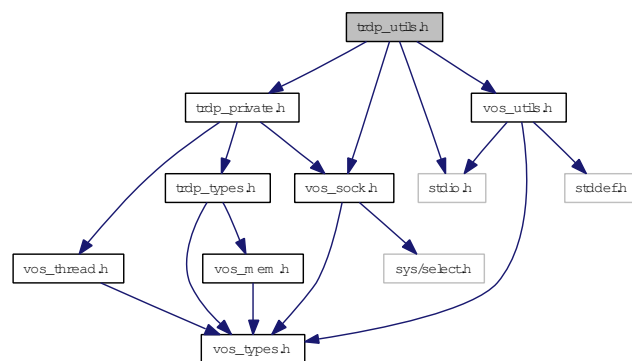


5.24 trdp_utils.h File Reference

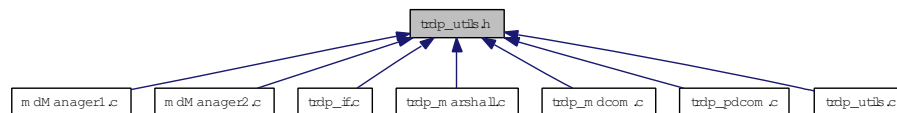
Common utilities for TRDP communication.

```
#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_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.
- `TRDP_ERR_T trdp_requestSocket` (`TRDP_SOCKETS_T iface[]`, const `TRDP_SEND_PARAM_T *params`, `TRDP_IP_ADDR_T srcIP`, `TRDP SOCK_TYPE_T usage`, `TRDP_OPTION_T options`, `BOOL rcvOnly`, `INT32 *pIndex`)
Handle the socket pool: Request a socket from our socket pool.
- `TRDP_ERR_T trdp_releaseSocket` (`TRDP_SOCKETS_T iface[]`, `INT32 index`)
Handle the socket pool: Release a socket from our socket pool.
- `UINT32 trdp_packetSizePD` (`UINT32 dataSize`)
Get the packet size from the raw data size.
- `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.

5.24.1 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](#) 138 2012-11-14 11:34:11Z 97031

5.24.2 Function Documentation**5.24.2.1 int am_big_endian ()**

Determine if we are Big or Little endian.

Return values:

!= 0 we are big endian

0 we are little endian

5.24.2.2 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:

**5.24.2.3 void trdp_initSockets (TRDP_SOCKETS_T *iface*[])**

Handle the socket pool: Initialize it.

Parameters:

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

5.24.2.4 **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:



5.24.2.5 **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

5.24.2.6 **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

5.24.2.7 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

5.24.2.8 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

5.24.2.9 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

5.24.2.10 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

5.24.2.11 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

5.24.2.12 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

5.24.2.13 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

5.24.2.14 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

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

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

Parameters:

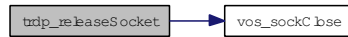
- ↔ *iface* socket pool
- ← *index* index of socket to release

Return values:

- TRDP_NO_ERR

TRDP_PARAM_ERR

Here is the call graph for this function:



5.24.2.16 **TRDP_ERR_T** trdp_requestSocket (TRDP_SOCKETS_T *iface*[],
const TRDP_SEND_PARAM_T * *params*, TRDP_IP_ADDR_T *srcIP*,
TRDP SOCK_TYPE_T *usage*, TRDP_OPTION_T *options*, BOOL *rcvOnly*, INT32 *
pIndex)

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

Parameters:

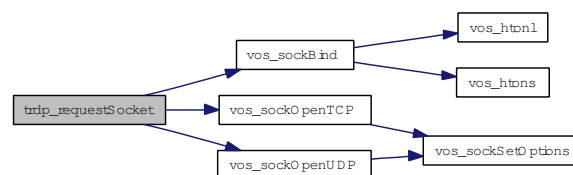
- ↔ *iface* socket pool
- ← *params* parameters to use
- ← *srcIP* IP to bind to (0 = any address)
- ← *usage* type and port to bind to
- ← *options* blocking/nonblocking
- ← *rcvOnly* only used for receiving
- *pIndex* returned index of socket pool

Return values:

TRDP_NO_ERR

TRDP_PARAM_ERR

Here is the call graph for this function:

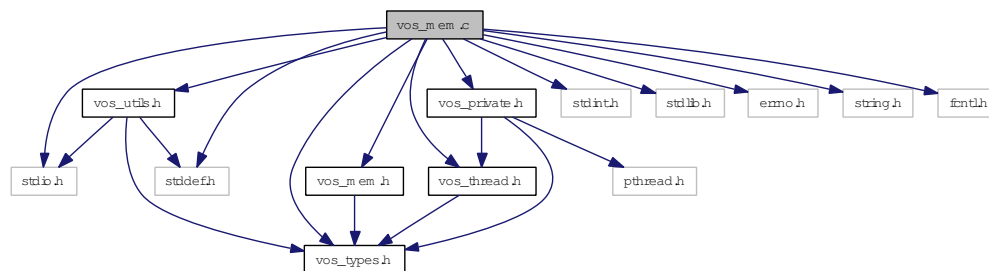


5.25 vos_mem.c File Reference

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

- EXT_DECL [VOS_ERR_T vos_memInit](#) (UINT8 *pMemoryArea, UINT32 size, const UINT32 fragMem[VOS_MEM_NBLOCKSIZES])
Initialize the memory unit.
- EXT_DECL [VOS_ERR_T vos_memDelete](#) (UINT8 *pMemoryArea)
Delete the memory area.
- EXT_DECL UINT8 * [vos_memAlloc](#) (UINT32 size)
Allocate a block of memory (from memory area above).
- EXT_DECL [VOS_ERR_T vos_memFree](#) (void *pMemBlock)
Deallocate a block of memory (from memory area above).
- EXT_DECL [VOS_ERR_T vos_memCount](#) (UINT32 *pAllocatedMemory, UINT32 *pFreeMemory, UINT32 *pMinFree, UINT32 *pNumAllocBlocks, UINT32 *pNumAllocErr,

UINT32 *pNumFreeErr, UINT32 allocBlockSize[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.

5.25.1 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](#) 150 2012-11-19 12:53:43Z bloehr

5.25.2 Function Documentation

5.25.2.1 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

5.25.2.2 EXT_DECL UINT8* vos_memAlloc (UINT32 size)

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

Parameters:

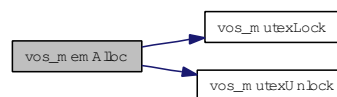
← *size* Size of requested block

Return values:

Pointer to memory area

NULL if no memory available

Here is the call graph for this function:


5.25.2.3 EXT_DECL VOS_ERR_T vos_memCount (UINT32 *pAllocatedMemory, UINT32 *pFreeMemory, UINT32 *pMinFree, UINT32 *pNumAllocBlocks, UINT32 *pNumAllocErr, UINT32 *pNumFreeErr, UINT32 allocBlockSize[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
- *allocBlockSize* Pointer to list of allocated memory blocks
- *usedBlockSize* Pointer to list of used memory blocks

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

5.25.2.4 EXT_DECL VOS_ERR_T vos_memDelete (UINT8 * *pMemoryArea*)

Delete the memory area.

This will eventually invalidate any previously allocated memory blocks! It should be called last before the application quits. No further access to the memory blocks is allowed after this call.

Parameters:

← *pMemoryArea* Pointer to memory area to use

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

Here is the call graph for this function:



5.25.2.5 EXT_DECL VOS_ERR_T vos_memFree (void * *pMemBlock*)

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

Parameters:

← *pMemBlock* Pointer to memory block to be freed

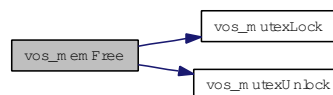
Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

Here is the call graph for this function:



5.25.2.6 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.

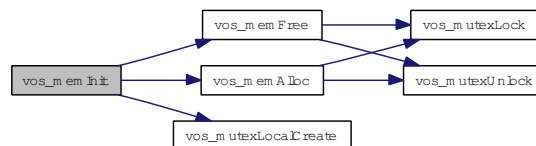
Parameters:

- ← *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:



5.25.2.7 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

Initialize the memory unit.

- EXT_DECL [VOS_ERR_T vos_memDelete](#) (UINT8 *pMemoryArea)
Delete the memory area.
- EXT_DECL UINT8 * [vos_memAlloc](#) (UINT32 size)
Allocate a block of memory (from memory area above).
- EXT_DECL [VOS_ERR_T vos_memFree](#) (void *pMemBlock)
Deallocate a block of memory (from memory area above).
- EXT_DECL [VOS_ERR_T vos_memCount](#) (UINT32 *pAllocatedMemory, UINT32 *pFreeMemory, UINT32 *pMinFree, UINT32 *pNumAllocBlocks, UINT32 *pNumAllocErr, UINT32 *pNumFreeErr, UINT32 allocBlockSize[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.

5.26.1 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](#) 149 2012-11-19 12:51:57Z bloehr

5.26.2 Define Documentation

5.26.2.1 #define VOS_MEM_BLOCKSIZES

Value:


```
{32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, \
    16384, 32768, 65536, 131072, 262144, 524288}
```

We internally allocate memory always by these block sizes.

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

5.26.2.2 #define VOS_MEM_PREALLOCATE {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 4, 0, 0}

Default pre-allocation of free memory blocks.

To avoid problems with too many small blocks and no large one. Specify how many of each block size that should be pre-allocated (and freed!) to pre-segment the memory area.

5.26.3 Function Documentation

5.26.3.1 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 qsort 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 values:

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

5.26.3.2 EXT_DECL UINT8* vos_memAlloc (UINT32 size)

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

Parameters:

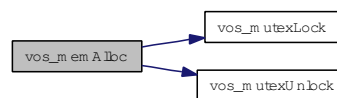
← *size* Size of requested block

Return values:

Pointer to memory area

NULL if no memory available

Here is the call graph for this function:



5.26.3.3 EXT_DECL VOS_ERR_T vos_memCount (UINT32 *pAllocatedMemory, UINT32 *pFreeMemory, UINT32 *pMinFree, UINT32 *pNumAllocBlocks, UINT32 *pNumAllocErr, UINT32 *pNumFreeErr, UINT32 allocBlockSize[VOS_MEM_NBBLOCKSIZES], UINT32 usedBlockSize[VOS_MEM_NBBLOCKSIZES])

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
- *allocBlockSize* Pointer to list of allocated memory blocks
- *usedBlockSize* Pointer to list of used memory blocks

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

5.26.3.4 EXT_DECL VOS_ERR_T vos_memDelete (UINT8 *pMemoryArea)

Delete the memory area.

This will eventually invalidate any previously allocated memory blocks! It should be called last before the application quits. No further access to the memory blocks is allowed after this call.

Parameters:

← *pMemoryArea* Pointer to memory area to use

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

This will eventually invalidate any previously allocated memory blocks! It should be called last before the application quits. No further access to the memory blocks is allowed after this call.

Parameters:

← *pMemoryArea* Pointer to memory area to use

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

Here is the call graph for this function:



5.26.3.5 EXT_DECL VOS_ERR_T vos_memFree (void * *pMemBlock*)

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

Parameters:

← *pMemBlock* Pointer to memory block to be freed

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

Parameters:

← *pMemBlock* Pointer to memory block to be freed

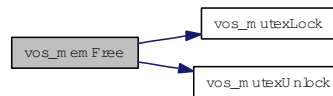
Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_PARAM_ERR parameter out of range/invalid

Here is the call graph for this function:



5.26.3.6 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.

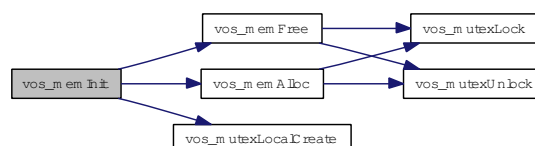
Parameters:

- ← *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:



5.26.3.7 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 values:

none 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 $\text{arg1} < \text{arg2}$, return 0 if $\text{arg1} == \text{arg2}$, return +n if $\text{arg1} > \text{arg2}$ where n is an integer != 0

Return values:

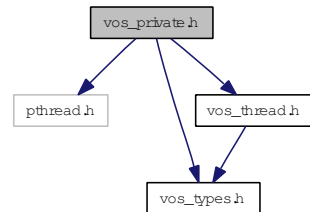
none

5.27 vos_private.h File Reference

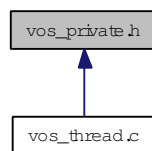
Private definitions for the OS abstraction layer.

```
#include <pthread.h>
#include "vos_types.h"
#include "vos_thread.h"
```

Include dependency graph for 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.
- [VOS_ERR_T vos_mutexLocalDelete](#) (struct VOS_MUTEX *pMutex)
Delete a mutex.

5.27.1 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 159 2012-11-20 16:51:12Z bloehr

5.27.2 Function Documentation**5.27.2.1 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

5.27.2.2 VOS_ERR_T vos_mutexLocalDelete (struct VOS_MUTEX * *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* Pointer to mutex struct

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

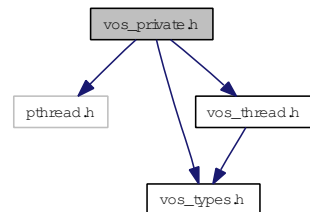
VOS_MUTEX_ERR no such mutex

5.28 vos_private.h File Reference

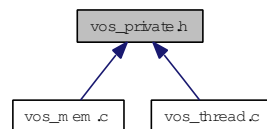
Private definitions for the OS abstraction layer.

```
#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.
- [VOS_ERR_T vos_mutexLocalDelete](#) (struct VOS_MUTEX *pMutex)
Delete a mutex.

5.28.1 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 159 2012-11-20 16:51:12Z bloehr

5.28.2 Function Documentation**5.28.2.1 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

5.28.2.2 VOS_ERR_T vos_mutexLocalDelete (struct VOS_MUTEX * *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* Pointer to mutex struct

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

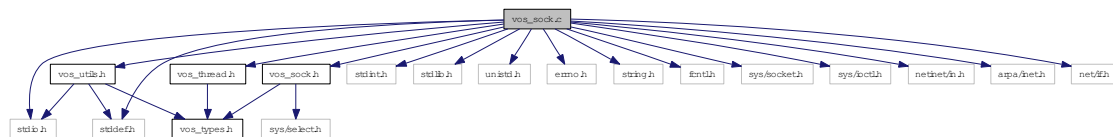
VOS_MUTEX_ERR no such mutex

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 <netinet/in.h>
#include <arpa/inet.h>
#include <net/if.h>
#include "vos_utils.h"
#include "vos_sock.h"
#include "vos_thread.h"
```

Include dependency graph for posix/vos_sock.c:



Functions

- EXT_DECL_UINT16 [vos_htons](#) (UINT16 val)
Byte swapping.
- EXT_DECL_UINT16 [vos_ntohs](#) (UINT16 val)
Byte swapping 2 Bytes.
- EXT_DECL_UINT32 [vos_htonl](#) (UINT32 val)
Byte swapping 4 Bytes.
- EXT_DECL_UINT32 [vos_ntohl](#) (UINT32 val)
Byte swapping 4 Bytes.
- EXT_DECL_BOOL [vos_isMulticast](#) (UINT32 ipAddress)
Check if the supplied address is a multicast group address.

- EXT_DECL UINT32 [vos_dottedIP](#) (const CHAR8 *pDottedIP)
Convert IP address.
- EXT_DECL const CHAR8 * [vos_ipDotted](#) (UINT32 ipAddress)
Convert IP address to dotted dec.
- EXT_DECL [VOS_ERR_T](#) [vos_sockInit](#) (void)
Initialize the socket library.
- EXT_DECL [VOS_ERR_T](#) [vos_sockGetMAC](#) (UINT8 pMAC[6])
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 size, UINT32 ipAddress, UINT16 port)
Send UDP data.
- EXT_DECL [VOS_ERR_T](#) [vos_sockReceiveUDP](#) (INT32 sock, UINT8 *pBuffer, INT32 *pSize, UINT32 *pIPAddr)
Receive UDP data.
- EXT_DECL [VOS_ERR_T](#) [vos_sockBind](#) (INT32 sock, UINT32 ipAddress, UINT16 port)
Bind a socket to an address and port.
- EXT_DECL [VOS_ERR_T](#) [vos_sockListen](#) (INT32 sock, UINT32 backlog)
Listen for incoming connections.

- EXT_DECL [VOS_ERR_T vos_sockAccept](#) (INT32 sock, INT32 *pSock, UINT32 *pIPAddress, UINT16 *pPort)
Accept an incoming TCP connection.
- EXT_DECL [VOS_ERR_T vos_sockConnect](#) (INT32 sock, UINT32 ipAddress, UINT16 port)
Open a TCP connection.
- EXT_DECL [VOS_ERR_T vos_sockSendTCP](#) (INT32 sock, const UINT8 *pBuffer, UINT32 size)
Send TCP data.
- EXT_DECL [VOS_ERR_T vos_sockReceiveTCP](#) (INT32 sock, UINT8 *pBuffer, INT32 *pSize)
Receive TCP data.

5.29.1 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.

Id

vos_sock.c 159 2012-11-20 16:51:12Z bloehr

5.29.2 Function Documentation

5.29.2.1 EXT_DECL UINT32 vos_dottedIP (const CHAR8 *pDottedIP)

Convert IP address.

Convert IP address from dotted dec.

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 UINT32 vos_htonl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.29.2.3 EXT_DECL UINT16 vos_htons (UINT16 *val*)

Byte swapping.

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.29.2.4 EXT_DECL const CHAR8* vos_ipDotted (UINT32 *ipAddress*)

Convert IP address to dotted dec.

Parameters:

← *ipAddress* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

5.29.2.5 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

5.29.2.6 EXT_DECL UINT32 vos_ntohl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.29.2.7 EXT_DECL UINT16 vos_ntohs (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.29.2.8 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

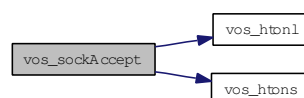
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR NULL parameter, parameter error

VOS_UNKNOWN_ERR socket descriptor unknown error

Here is the call graph for this function:



5.29.2.9 EXT_DECL VOS_ERR_T vos_sockBind (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Bind a socket to an address and port.

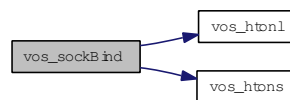
Parameters:

- ← *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:

**5.29.2.10 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

5.29.2.11 EXT_DECL VOS_ERR_T vos_sockConnect (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

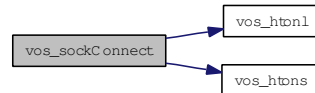
Open a TCP connection.

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:

**5.29.2.12 EXT_DECL VOS_ERR_T vos_sockGetMAC (UINT8 *pMAC*[6])**

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**5.29.2.13 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**5.29.2.14 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

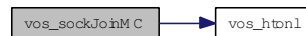
Return values:

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:



5.29.2.15 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

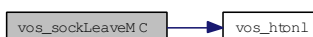
Return values:

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:



5.29.2.16 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

5.29.2.17 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:



5.29.2.18 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:



5.29.2.19 EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 *sock*, UINT8 * *pBuffer*, INT32 * *pSize*)

Receive TCP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error 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

5.29.2.20 EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 *sock*, UINT8 * *pBuffer*, INT32 * *pSize*, UINT32 * *pIPAddr*)

Receive UDP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error 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
- *pIPAddr* source IP

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be read
- VOS_NODATA_ERR** no data in non-blocking

Here is the call graph for this function:



5.29.2.21 EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 *size*)

Send TCP data.

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ← *size* size of the data to send

Return values:

- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown, parameter error
- VOS_IO_ERR* data could not be sent

5.29.2.22 EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 *size*, UINT32 *ipAddress*, UINT16 *port*)

Send UDP data.

Send data to the supplied address and port.

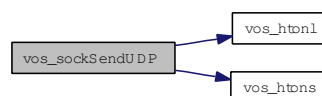
Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ← *size* size of the data to send
- ← *ipAddress* destination IP
- ← *port* destination port

Return values:

- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown, parameter error
- VOS_IO_ERR* data could not be sent
- VOS_MEM_ERR* resource error

Here is the call graph for this function:



5.29.2.23 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

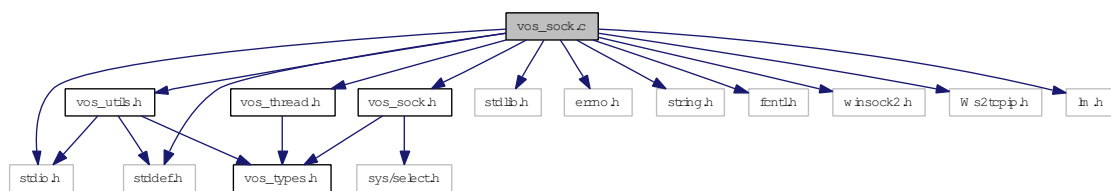
VOS_PARAM_ERR sock descriptor unknown

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 <lm.h>
#include "vos_utils.h"
#include "vos_sock.h"
#include "vos_thread.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 BOOL [vos_isMulticast](#) (UINT32 ipAddress)
Check if the supplied address is a multicast group address.
- EXT_DECL UINT32 [vos_dottedIP](#) (const CHAR8 *pDottedIP)
Convert IP address.

- EXT_DECL const CHAR8 * [vos_ipDotted](#) (UINT32 ipAddress)
Convert IP address to dotted dec.
- EXT_DECL [VOS_ERR_T](#) [vos_sockInit](#) (void)
Initialize the socket library.
- EXT_DECL [VOS_ERR_T](#) [vos_sockGetMAC](#) (UINT8 pMAC[6])
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 size, UINT32 ipAddress, UINT16 port)
Send UDP data.
- EXT_DECL [VOS_ERR_T](#) [vos_sockReceiveUDP](#) (INT32 sock, UINT8 *pBuffer, INT32 *pSize, UINT32 *pIPAddr)
Receive UDP data.
- EXT_DECL [VOS_ERR_T](#) [vos_sockBind](#) (INT32 sock, UINT32 ipAddress, UINT16 port)
Bind a socket to an address and port.
- EXT_DECL [VOS_ERR_T](#) [vos_sockListen](#) (INT32 sock, UINT32 backlog)
Listen for incoming connections.
- EXT_DECL [VOS_ERR_T](#) [vos_sockAccept](#) (INT32 sock, INT32 *pSock, UINT32 *pIPAddress, UINT16 *pPort)
Accept an incoming TCP connection.

- EXT_DECL [VOS_ERR_T vos_sockConnect](#) (INT32 sock, UINT32 ipAddress, UINT16 port)
Open a TCP connection.
- EXT_DECL [VOS_ERR_T vos_sockSendTCP](#) (INT32 sock, const UINT8 *pBuffer, UINT32 size)
Send TCP data.
- EXT_DECL [VOS_ERR_T vos_sockReceiveTCP](#) (INT32 sock, UINT8 *pBuffer, INT32 *pSize)
Receive TCP data.

5.30.1 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 159 2012-11-20 16:51:12Z bloehr

5.30.2 Function Documentation

5.30.2.1 EXT_DECL UINT32 vos_dottedIP (const CHAR8 *pDottedIP)

Convert IP address.

Convert IP address from dotted dec.

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 UINT32 vos_htonl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.30.2.3 EXT_DECL UINT16 vos_htons (UINT16 *val*)

Byte swapping.

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.30.2.4 EXT_DECL const CHAR8* vos_ipDotted (UINT32 *ipAddress*)

Convert IP address to dotted dec.

Parameters:

← *ipAddress* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

5.30.2.5 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

5.30.2.6 EXT_DECL UINT32 vos_ntohl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.30.2.7 EXT_DECL UINT16 vos_ntohs (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.30.2.8 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.

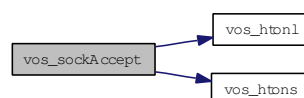
Parameters:

- ← *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** socket descriptor unknown error

Here is the call graph for this function:



5.30.2.9 EXT_DECL VOS_ERR_T vos_sockBind (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Bind a socket to an address and port.

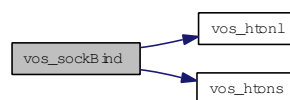
Parameters:

- ← *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:

**5.30.2.10 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

5.30.2.11 EXT_DECL VOS_ERR_T vos_sockConnect (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

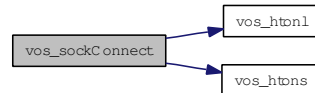
Open a TCP connection.

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:



5.30.2.12 EXT_DECL VOS_ERR_T vos_sockGetMAC (UINT8 *pMAC*[6])

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

5.30.2.13 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

5.30.2.14 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

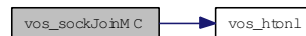
Return values:

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:



5.30.2.15 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

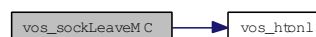
Return values:

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:



5.30.2.16 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

5.30.2.17 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:



5.30.2.18 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:



5.30.2.19 EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 *sock*, UINT8 * *pBuffer*, INT32 * *pSize*)

Receive TCP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error 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

5.30.2.20 EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 *sock*, UINT8 * *pBuffer*, INT32 * *pSize*, UINT32 * *pIPAddr*)

Receive UDP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error 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
- *pIPAddr* source IP

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be read
- VOS_NODATA_ERR** no data in non-blocking

Here is the call graph for this function:



5.30.2.21 EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 *size*)

Send TCP data.

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ← *size* size of the data to send

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be sent

5.30.2.22 EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 *size*, UINT32 *ipAddress*, UINT16 *port*)

Send UDP data.

Send data to the supplied address and port.

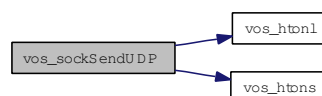
Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ← *size* size of the data to send
- ← *ipAddress* destination IP
- ← *port* destination port

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be sent
- VOS_MEM_ERR** resource error

Here is the call graph for this function:



5.30.2.23 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

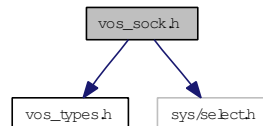
VOS_PARAM_ERR sock descriptor unknown

5.31 vos_sock.h File Reference

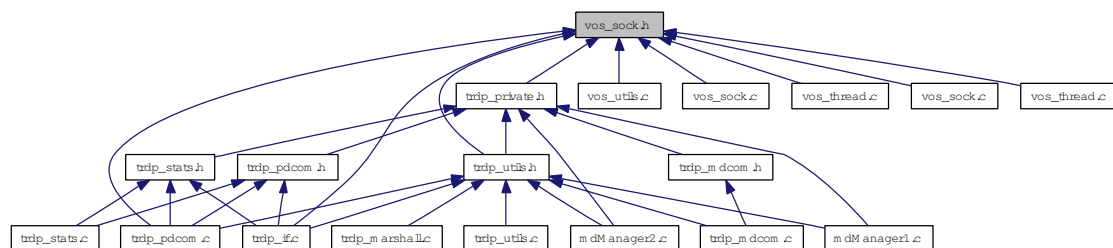
Typedefs for OS abstraction.

```
#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_SOCKET_OPT_T](#)
Common socket options.

Defines

- #define [VOS_MAX_SOCKET_CNT](#) 80
The maximum number of concurrent usable sockets.
- #define [VOS_TTL_MULTICAST](#) 64
The maximum hops a multicast packet can go.

Functions

- EXT_DECL UINT16 [vos_htons](#) (UINT16 val)
Byte swapping 2 Bytes.
- EXT_DECL UINT16 [vos_ntohs](#) (UINT16 val)
Byte swapping 2 Bytes.

- EXT_DECL UINT32 [vos_htonl](#) (UINT32 val)
Byte swapping 4 Bytes.
- EXT_DECL UINT32 [vos_ntohl](#) (UINT32 val)
Byte swapping 4 Bytes.
- EXT_DECL BOOL [vos_isMulticast](#) (UINT32 ipAddress)
Check if the supplied address is a multicast group address.
- EXT_DECL 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 [VOS_ERR_T](#) [vos_sockInit](#) (void)
Initialize the socket library.
- EXT_DECL [VOS_ERR_T](#) [vos_sockGetMAC](#) (UINT8 pMAC[6])
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 size, UINT32 ipAddress, UINT16 port)
Send UDP data.
- EXT_DECL [VOS_ERR_T](#) [vos_sockReceiveUDP](#) (INT32 sock, UINT8 *pBuffer, INT32 *pSize, UINT32 *pIPAddr)

Receive UDP data.

- EXT_DECL [VOS_ERR_T vos_sockBind](#) (INT32 sock, UINT32 ipAddress, UINT16 port)
Bind a socket to an address and port.
- EXT_DECL [VOS_ERR_T vos_sockListen](#) (INT32 sock, UINT32 backlog)
Listen for incoming TCP connections.
- EXT_DECL [VOS_ERR_T vos_sockAccept](#) (INT32 sock, INT32 *pSock, UINT32 *pIPAddress, UINT16 *pPort)
Accept an incoming TCP connection.
- EXT_DECL [VOS_ERR_T vos_sockConnect](#) (INT32 sock, UINT32 ipAddress, UINT16 port)
Open a TCP connection.
- EXT_DECL [VOS_ERR_T vos_sockSendTCP](#) (INT32 sock, const UINT8 *pBuffer, UINT32 size)
Send TCP data.
- EXT_DECL [VOS_ERR_T vos_sockReceiveTCP](#) (INT32 sock, UINT8 *pBuffer, INT32 *pSize)
Receive TCP data.

5.31.1 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](#) 148 2012-11-19 12:46:34Z bloehr

5.31.2 Function Documentation

5.31.2.1 EXT_DECL UINT32 vos_dottedIP (const CHAR8 * *pDottedIP*)

Convert IP address from dotted dec.

to !host! endianess

Parameters:

← *pDottedIP* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

Convert IP address from dotted dec.

Parameters:

← *pDottedIP* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

Here is the call graph for this function:

**5.31.2.2 EXT_DECL UINT32 vos_htonl (UINT32 val)**

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.31.2.3 EXT_DECL UINT16 vos_htons (UINT16 val)

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.31.2.4 EXT_DECL const CHAR8* vos_ipDotted (UINT32 *ipAddress*)

Convert IP address to dotted dec.
from !host! endianness

Parameters:

← *ipAddress* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

Parameters:

← *ipAddress* IP address as dotted decimal.

Return values:

address in UINT32 in host endianness

5.31.2.5 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

Parameters:

← *ipAddress* IP address to check.

Return values:

TRUE address is multicast

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

5.31.2.6 EXT_DECL UINT32 vos_ntohl (UINT32 *val*)

Byte swapping 4 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.31.2.7 EXT_DECL UINT16 vos_ntohs (UINT16 *val*)

Byte swapping 2 Bytes.

Parameters:

← *val* Initial value.

Return values:

swapped value

5.31.2.8 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.

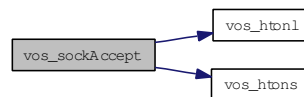
Parameters:

- ← *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:



5.31.2.9 EXT_DECL VOS_ERR_T vos_sockBind (INT32 sock, UINT32 ipAddress, UINT16 port)

Bind a socket to an address and port.

Parameters:

- ← *sock* socket descriptor
- ← *ipAddress* source IP to receive from, 0 for any
- ← *port* port to receive from

Return values:

- VOS_NO_ERR** no error
- VOS_INIT_ERR** module not initialised
- VOS_NOINIT_ERR** invalid handle
- VOS_PARAM_ERR** parameter out of range/invalid
- VOS_IO_ERR** Input/Output error

VOS_MEM_ERR resource error

Parameters:

- ← *sock* socket descriptor
- ← *ipAddress* source IP to receive on, 0 for any
- ← *port* port to receive on, 20548 for PD

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** Input/Output error
- VOS_MEM_ERR** resource error

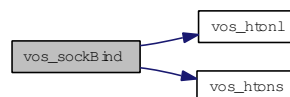
Parameters:

- ← *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:



5.31.2.10 EXT_DECL VOS_ERR_T vos_sockClose (INT32 *sock*)

Close a socket.

Release any resources aquired by this socket

Parameters:

- ← *sock* socket descriptor

Return values:

- VOS_NO_ERR** no error
- VOS_INIT_ERR** module not initialised
- VOS_NOINIT_ERR** invalid handle

Release any resources aquired by this socket

Parameters:

← *sock* socket descriptor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown

Release any resources aquired by this socket

Parameters:

← *sock* socket descriptor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown

5.31.2.11 EXT_DECL VOS_ERR_T vos_sockConnect (INT32 *sock*, UINT32 *ipAddress*, UINT16 *port*)

Open a TCP connection.

Parameters:

← *sock* socket descriptor

← *ipAddress* destination IP

← *port* destination port

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

Parameters:

← *sock* socket descriptor

← *ipAddress* destination IP

← *port* destination port

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR Input/Output error

VOS_MEM_ERR resource error

Parameters:

← *sock* socket descriptor

← *ipAddress* destination IP

← *port* destination port

Return values:

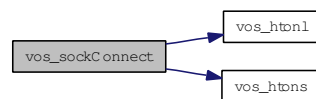
VOS_NO_ERR no error

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:



5.31.2.12 EXT_DECL VOS_ERR_T vos_sockGetMAC (UINT8 pMAC[6])

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

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

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

5.31.2.13 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

5.31.2.14 EXT_DECL VOS_ERR_T vos_sockJoinMC (INT32 sock, UINT32 mcAddress, UINT32 ipAddress)

Join a multicast group.

Note: Some target systems might not support this option.

Parameters:

← *sock* socket descriptor

← *mcAddress* multicast group to join

← *ipAddress* depicts interface on which to join, default 0 for any

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS SOCK_ERR option not supported

Note: Some targeted systems might not support this option.

Parameters:

← *sock* socket descriptor

← *mcAddress* multicast group to join

← *ipAddress* depicts interface on which to join, default 0 for any

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS SOCK_ERR option not supported

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:



5.31.2.15 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

Note: Some targeted systems might not support this option.

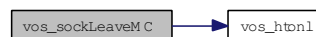
Parameters:

- ← *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:



5.31.2.16 EXT_DECL VOS_ERR_T vos_sockListen (INT32 *sock*, UINT32 *backlog*)

Listen for incoming TCP connections.

Parameters:

- ← *sock* socket descriptor
- ← *backlog* maximum connection attempts if system is busy

Return values:

- VOS_NO_ERR** no error
- VOS_INIT_ERR** module not initialised
- VOS_NOINIT_ERR** invalid handle
- VOS_PARAM_ERR** parameter out of range/invalid
- VOS_IO_ERR** Input/Output error
- VOS_MEM_ERR** resource error

Listen for incoming TCP connections.

Parameters:

- ← *sock* socket descriptor
- ← *backlog* maximum connection attempts if system is busy

Return values:

- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown, parameter error
- VOS_IO_ERR* Input/Output error
- VOS_MEM_ERR* resource error

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

5.31.2.17 EXT_DECL VOS_ERR_T vos_sockOpenTCP (INT32 * *pSock*, const VOS_SOCK_OPT_T * *pOptions*)

Create a TCP socket.

Return a socket descriptor for further calls. The socket options are optional and can be applied later.

Parameters:

- *pSock* pointer to socket descriptor returned
- ← *pOptions* pointer to socket options (optional)

Return values:

- VOS_NO_ERR* no error
- VOS_INIT_ERR* module not initialised
- VOS_PARAM_ERR* pSock == NULL
- VOS_SOCK_ERR* socket not available or option not supported

Return a socket descriptor for further calls. The socket options are optional and can be applied later.

Parameters:

- *pSock* pointer to socket descriptor returned
- ← *pOptions* pointer to socket options (optional)

Return values:*VOS_NO_ERR* no error*VOS_PARAM_ERR* pSock == NULL*VOS SOCK_ERR* socket not available or option not supported

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:



5.31.2.18 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:



5.31.2.19 EXT_DECL VOS_ERR_T vos_sockReceiveTCP (INT32 sock, UINT8 *pBuffer, INT32 *pSize)

Receive TCP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error 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_INIT_ERR** module not initialised**VOS_NOINIT_ERR** invalid handle**VOS_PARAM_ERR** parameter out of range/invalid**VOS_IO_ERR** data could not be read**VOS_MEM_ERR** resource error

VOS_NODATA_ERR no data in non-blocking

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error 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

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

5.31.2.20 EXT_DECL VOS_ERR_T vos_sockReceiveUDP (INT32 *sock*, UINT8 * *pBuffer*, INT32 * *pSize*, UINT32 * *pIPAddr*)

Receive UDP data.

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error 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
- *pIPAddr* source IP

Return values:

- VOS_NO_ERR** no error
- VOS_INIT_ERR** module not initialised
- VOS_NOINIT_ERR** invalid handle
- VOS_PARAM_ERR** parameter out of range/invalid
- VOS_IO_ERR** data could not be read
- VOS_MEM_ERR** resource error
- VOS_NODATA_ERR** no data in non-blocking

The caller must provide a sufficient sized buffer. If the supplied buffer is smaller than the bytes received, *pSize will reflect the number of copied bytes and the call should be repeated until *pSize is 0 (zero). If the socket was created in blocking-mode (default), then this call will block and will only return if data has been received or the socket was closed or an error 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
- *pIPAddr* source IP

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** sock descriptor unknown, parameter error
- VOS_IO_ERR** data could not be read
- VOS_NODATA_ERR** no data in non-blocking

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
- *pIPAddr* source IP

Return values:**VOS_NO_ERR** no error**VOS_PARAM_ERR** sock descriptor unknown, parameter error**VOS_IO_ERR** data could not be read**VOS_NODATA_ERR** no data in non-blocking

Here is the call graph for this function:



5.31.2.21 EXT_DECL VOS_ERR_T vos_sockSendTCP (INT32 *sock*, const UINT8 * *pBuffer*, UINT32 *size*)

Send TCP data.

Send data to the given socket.

Parameters:← *sock* socket descriptor← *pBuffer* pointer to data to send← *size* size of the data to send**Return values:****VOS_NO_ERR** no error**VOS_INIT_ERR** module not initialised**VOS_NOINIT_ERR** invalid handle**VOS_PARAM_ERR** parameter out of range/invalid**VOS_IO_ERR** data could not be sent**VOS_MEM_ERR** resource error

Send data to the supplied address and port.

Parameters:← *sock* socket descriptor← *pBuffer* pointer to data to send← *size* size of the data to send**Return values:****VOS_NO_ERR** no error**VOS_PARAM_ERR** sock descriptor unknown, parameter error**VOS_IO_ERR** data could not be sent

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ← *size* size of the data to send

Return values:

- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown, parameter error
- VOS_IO_ERR* data could not be sent

**5.31.2.22 EXT_DECL VOS_ERR_T vos_sockSendUDP (INT32 *sock*, const UINT8 * *pBuffer*,
UINT32 *size*, UINT32 *ipAddress*, UINT16 *port*)**

Send UDP data.

Send data to the given address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ← *size* size of the data to send
- ← *ipAddress* destination IP
- ← *port* destination port

Return values:

- VOS_NO_ERR* no error
- VOS_INIT_ERR* module not initialised
- VOS_NOINIT_ERR* invalid handle
- VOS_PARAM_ERR* parameter out of range/invalid
- VOS_IO_ERR* data could not be sent
- VOS_MEM_ERR* resource error

Send data to the supplied address and port.

Parameters:

- ← *sock* socket descriptor
- ← *pBuffer* pointer to data to send
- ← *size* size of the data to send
- ← *ipAddress* destination IP
- ← *port* destination port

Return values:

- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown, parameter error

VOS_IO_ERR data could not be sent

VOS_MEM_ERR resource error

Send data to the supplied address and port.

Parameters:

← **sock** socket descriptor

← **pBuffer** pointer to data to send

← **size** size of the data to send

← **ipAddress** destination IP

← **port** destination port

Return values:

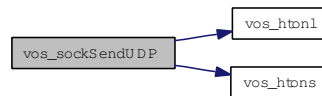
VOS_NO_ERR no error

VOS_PARAM_ERR sock descriptor unknown, parameter error

VOS_IO_ERR data could not be sent

VOS_MEM_ERR resource error

Here is the call graph for this function:



5.31.2.23 EXT_DECL VOS_ERR_T vos_sockSetOptions (INT32 *sock*, const VOS_SOCK_OPT_T **pOptions*)

Set socket options.

Note: Some target systems might not support each option.

Parameters:

← **sock** socket descriptor

← **pOptions** pointer to socket options (optional)

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_PARAM_ERR parameter out of range/invalid

VOS_SOCK_ERR socket not available or option not supported

Note: Some targeted systems might not support every option.

Parameters:

- ← *sock* socket descriptor
- ← *pOptions* pointer to socket options (optional)

Return values:

- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown

Note: Some targeted systems might not support every option.

Parameters:

- ← *sock* socket descriptor
- ← *pOptions* pointer to socket options (optional)

Return values:

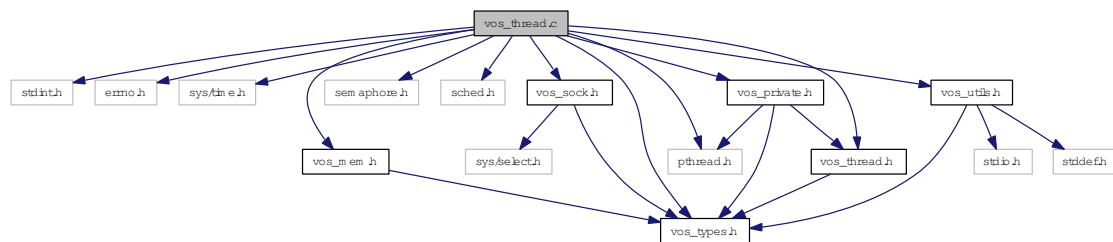
- VOS_NO_ERR* no error
- VOS_PARAM_ERR* sock descriptor unknown

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 `VOS_ERR_T vos_getTime` (VOS_TIME_T *pTime)
Return the current time in sec and us.
- EXT_DECL `const CHAR8 * vos_getTimeStamp` (void)
Get a time-stamp string.
- EXT_DECL `VOS_ERR_T vos_clearTime` (VOS_TIME_T *pTime)
Clear the time stamp.
- EXT_DECL `VOS_ERR_T vos_addTime` (VOS_TIME_T *pTime, const VOS_TIME_T *pAdd)
Add the second to the first time stamp, return sum in first.
- EXT_DECL `VOS_ERR_T vos_subTime` (VOS_TIME_T *pTime, const VOS_TIME_T *pSub)
Subtract the second from the first time stamp, return diff in first.
- EXT_DECL `VOS_ERR_T vos_divTime` (VOS_TIME_T *pTime, UINT32 div)
Divide the first time value by the second, return quotient in first.
- EXT_DECL `VOS_ERR_T 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 `VOS_ERR_T vos_getUuid` (VOS_UUID_T pUuid)
Get a universal unique identifier according to RFC 4122 time based version.
- EXT_DECL `VOS_ERR_T vos_mutexCreate` (VOS_MUTEX_T *pMutex)
Create a recursive mutex.
- EXT_DECL `VOS_ERR_T vos_mutexLocalCreate` (struct VOS_MUTEX *pMutex)
Create a recursive mutex.
- EXT_DECL `VOS_ERR_T vos_mutexDelete` (VOS_MUTEX_T pMutex)
Delete a mutex.
- EXT_DECL `VOS_ERR_T 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 [VOS_ERR_T](#) [vos_semaDelete](#) ([VOS_SEMA_T](#) sema)

Delete a semaphore.

- EXT_DECL [VOS_ERR_T](#) [vos_semaTake](#) ([VOS_SEMA_T](#) sema, UINT32 timeout)

Take a semaphore.

- EXT_DECL [VOS_ERR_T](#) [vos_semaGive](#) ([VOS_SEMA_T](#) sema)

Give a semaphore.

5.32.1 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 159 2012-11-20 16:51:12Z bloehr

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 VOS_ERR_T vos_addTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pAdd*)

Add the second to the first time stamp, return sum in first.

Parameters:

↔ *pTime* Pointer to time value

← *pAdd* Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.32.2.3 EXT_DECL VOS_ERR_T vos_clearTime (VOS_TIME_T * *pTime*)

Clear the time stamp.

Parameters:

→ *pTime* Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.32.2.4 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 VOS_ERR_T vos_divTime (VOS_TIME_T * *pTime*, UINT32 *div*)

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

← *div* Divisor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.32.2.6 EXT_DECL VOS_ERR_T vos_getTime (VOS_TIME_T * *pTime*)

Return the current time in sec and us.

Parameters:

→ *pTime* Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

5.32.2.7 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 VOS_ERR_T vos_getUuid (VOS_UUID_T *pUUID*)

Get a universal unique identifier according to RFC 4122 time based version.

Parameters:

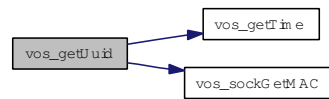
→ *pUUID* Pointer to a universal unique identifier

Return values:

VOS_NO_ERR no error

VOS_UNKNOWN_ERR Could not create UUID

Here is the call graph for this function:



5.32.2.9 EXT_DECL VOS_ERR_T 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

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

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:

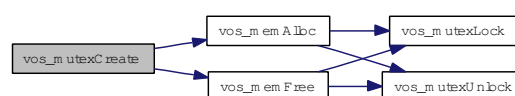
VOS_NO_ERR no error

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 VOS_ERR_T vos_mutexDelete (VOS_MUTEX_T *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* mutex handle

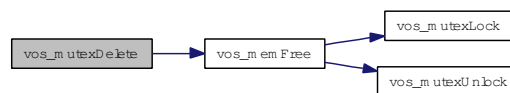
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR no such mutex

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 VOS_ERR_T vos_mutexLocalDelete (struct VOS_MUTEX * *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* Pointer to mutex struct

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR no such mutex

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

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR no such mutex

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

5.32.2.18 EXT_DECL VOS_ERR_T vos_semaDelete (VOS_SEMA_T *sema*)

Delete a semaphore.

This will eventually release any processes waiting for the semaphore.

Parameters:

- ← *sema* semaphore handle

Return values:

- VOS_NO_ERR* no error
- VOS_INIT_ERR* module not initialised
- VOS_NOINIT_ERR* invalid handle

5.32.2.19 EXT_DECL VOS_ERR_T vos_semaGive (VOS_SEMA_T *sema*)

Give a semaphore.

Release (increase) a semaphore.

Parameters:

- ← *sema* semaphore handle

Return values:

- VOS_NO_ERR* no error
- VOS_INIT_ERR* module not initialised
- VOS_NOINIT_ERR* invalid handle
- VOS_SEM_ERR* could not release semaphore

5.32.2.20 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 forever

Return values:

- VOS_NO_ERR* no error
- VOS_INIT_ERR* module not initialised
- VOS_NOINIT_ERR* invalid handle
- VOS_PARAM_ERR* parameter out of range/invalid
- VOS_SEMA_ERR* could not get semaphore in time

5.32.2.21 EXT_DECL VOS_ERR_T vos_subTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pSub*)

Subtract the second from the first time stamp, return diff in first.

Parameters:

- ↔ *pTime* Pointer to time value
- ← *pSub* Pointer to time value

Return values:

- VOS_NO_ERR* no error
- VOS_PARAM_ERR* parameter must not be NULL

5.32.2.22 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

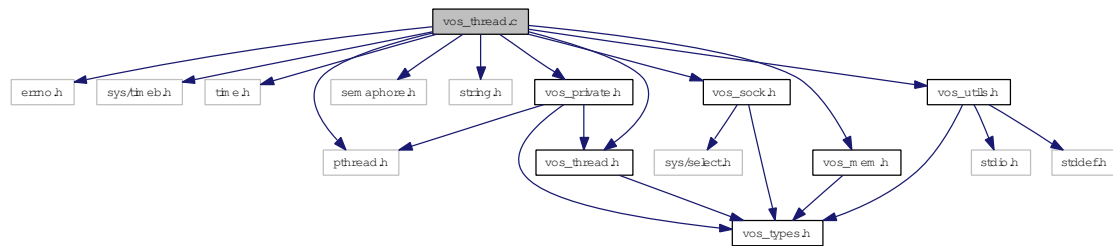
VOS_THREAD_ERR cancel failed

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 [VOS_ERR_T vos_getTime](#) (VOS_TIME_T *pTime)
Return the current time in sec and us.
- EXT_DECL const CHAR8 * [vos_getTimeStamp](#) (void)
Get a time-stamp string.
- EXT_DECL [VOS_ERR_T vos_clearTime](#) (VOS_TIME_T *pTime)
Clear the time stamp.
- EXT_DECL [VOS_ERR_T vos_addTime](#) (VOS_TIME_T *pTime, const VOS_TIME_T *pAdd)
Add the second to the first time stamp, return sum in first.
- EXT_DECL [VOS_ERR_T vos_subTime](#) (VOS_TIME_T *pTime, const VOS_TIME_T *pSub)
Subtract the second from the first time stamp, return diff in first.
- EXT_DECL [VOS_ERR_T vos_divTime](#) (VOS_TIME_T *pTime, UINT32 div)
Divide the first time value by the second, return quotient in first.
- EXT_DECL [VOS_ERR_T 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 [VOS_ERR_T vos_getUuid](#) (VOS_UUID_T pUulD)
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 [VOS_ERR_T vos_mutexDelete](#) (VOS_MUTEX_T pMutex)
Delete a mutex.
- EXT_DECL [VOS_ERR_T 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.

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, 2012. [vos_thread.c](#) uses pthreads-w32 (<http://sourceware.org/pthreads-win32/>) under LGPL license

Id

[vos_thread.c](#) 159 2012-11-20 16:51:12Z bloehr

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 VOS_ERR_T vos_addTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pAdd*)

Add the second to the first time stamp, return sum in first.

Parameters:

↔ *pTime* Pointer to time value

← *pAdd* Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.33.2.3 EXT_DECL VOS_ERR_T vos_clearTime (VOS_TIME_T * *pTime*)

Clear the time stamp.

Parameters:

→ *pTime* Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.33.2.4 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 VOS_ERR_T vos_divTime (VOS_TIME_T * *pTime*, UINT32 *div*)

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

← *div* Divisor

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

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 VOS_ERR_T vos_getTime (VOS_TIME_T * pTime)

Return the current time in sec and us.

Parameters:

→ *pTime* Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter out of range/invalid

5.33.2.8 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 VOS_ERR_T vos_getUuid (VOS_UUID_T pUUID)

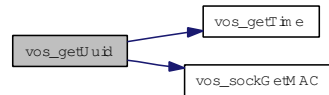
Get a universal unique identifier according to RFC 4122 time based version.

Parameters:

→ *pUUID* Pointer to a universal unique identifier

Return values:**VOS_NO_ERR** no error**VOS_INIT_ERR** module not initialised

Here is the call graph for this function:

**5.33.2.10 EXT_DECL VOS_ERR_T 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**Return values:****VOS_NO_ERR** no error**VOS_PARAM_ERR** parameter must not be NULL**5.33.2.11 EXT_DECL VOS_ERR_T vos_mutexCreate (VOS_MUTEX_T *pMutex)**

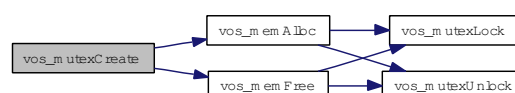
Create a recursive mutex.

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 VOS_ERR_T vos_mutexDelete (VOS_MUTEX_T *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* mutex handle

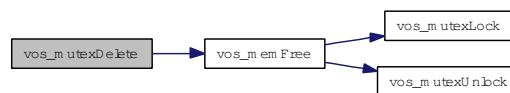
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR no such mutex

Here is the call graph for this function:



5.33.2.13 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

5.33.2.14 EXT_DECL VOS_ERR_T vos_mutexLocalDelete (struct VOS_MUTEX * *pMutex*)

Delete a mutex.

Release the resources taken by the mutex.

Parameters:

← *pMutex* Pointer to mutex struct

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR no such mutex

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

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR *pMutex* == NULL or wrong type

VOS_MUTEX_ERR no such mutex

5.33.2.18 EXT_DECL VOS_ERR_T vos_subTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pSub*)

Subtract the second from the first time stamp, return diff in first.

Parameters:

- ↔ *pTime* Pointer to time value
- ← *pSub* Pointer to time value

Return values:

- VOS_NO_ERR** no error
- VOS_PARAM_ERR** parameter must not be NULL

5.33.2.19 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
- VOS_THREAD_ERR** thread creation error
- VOS_INIT_ERR** no threads available

Here is the call graph for this function:



5.33.2.20 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_PARAM_ERR parameter out of range/invalid

5.33.2.21 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

5.33.2.22 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_PARAM_ERR parameter out of range/invalid

5.33.2.23 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

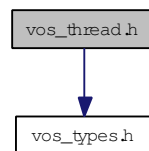
VOS_THREAD_ERR cancel failed

5.34 vos_thread.h File Reference

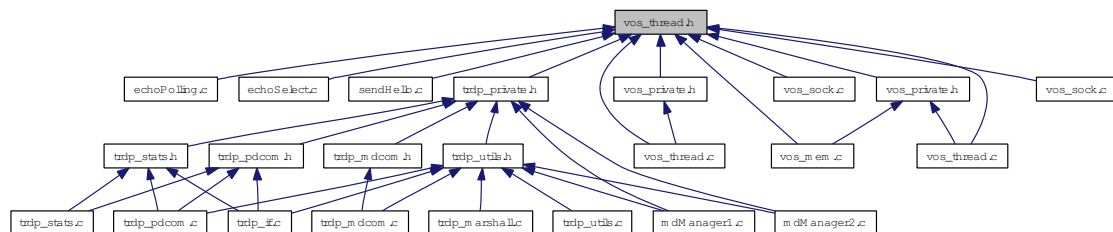
Threading functions for OS abstraction.

```
#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.

Typedefs

- `typedef uint8_t 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 [VOS_ERR_T](#) [vos_getTime](#) ([VOS_TIME_T](#) *pTime)
Return the current time in sec and us.
- EXT_DECL const [CHAR8](#) * [vos_getTimeStamp](#) (void)
Get a time-stamp string.
- EXT_DECL [VOS_ERR_T](#) [vos_clearTime](#) ([VOS_TIME_T](#) *pTime)
Clear the time stamp.
- EXT_DECL [VOS_ERR_T](#) [vos_addTime](#) ([VOS_TIME_T](#) *pTime, const [VOS_TIME_T](#) *pAdd)
Add the second to the first time stamp, return sum in first.
- EXT_DECL [VOS_ERR_T](#) [vos_subTime](#) ([VOS_TIME_T](#) *pTime, const [VOS_TIME_T](#) *pSub)
Subtract the second from the first time stamp, return diff in first.
- EXT_DECL [INT32](#) [vos_cmpTime](#) (const [VOS_TIME_T](#) *pTime, const [VOS_TIME_T](#) *pCmp)
Compare the second from the first time stamp, return diff in first.
- EXT_DECL [VOS_ERR_T](#) [vos_divTime](#) ([VOS_TIME_T](#) *pTime, [UINT32](#) div)
Divide the first time by the second, return quotient in first.

- EXT_DECL [VOS_ERR_T vos_mulTime](#) ([VOS_TIME_T](#) *pTime, UINT32 mul)
Multiply the first time by the second, return product in first.
- EXT_DECL [VOS_ERR_T vos_getUuid](#) ([VOS_UUID_T](#) pUulD)
Get a universal unique identifier according to RFC 4122 time based version.
- EXT_DECL [VOS_ERR_T vos_mutexCreate](#) ([VOS_MUTEX_T](#) *pMutex)
Create a mutex.
- EXT_DECL [VOS_ERR_T vos_mutexDelete](#) ([VOS_MUTEX_T](#) 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.

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](#) 159 2012-11-20 16:51:12Z bloehr

5.34.2 Function Documentation

5.34.2.1 EXT_DECL [VOS_ERR_T vos_addTime](#) ([VOS_TIME_T](#) *pTime, const [VOS_TIME_T](#) *pAdd)

Add the second to the first time stamp, return sum in first.

Parameters:↔ *pTime* Pointer to time value← *pAdd* Pointer to time value**Return values:***VOS_NO_ERR* no error*VOS_PARAM_ERR* parameter must not be NULL**Parameters:**↔ *pTime* Pointer to time value← *pAdd* Pointer to time value**Return values:***VOS_NO_ERR* no error*VOS_PARAM_ERR* parameter must not be NULL**5.34.2.2 EXT_DECL VOS_ERR_T vos_clearTime (VOS_TIME_T * *pTime*)**

Clear the time stamp.

Parameters:→ *pTime* Pointer to time value**Return values:***VOS_NO_ERR* no error*VOS_PARAM_ERR* parameter must not be NULL**Parameters:**→ *pTime* Pointer to time value**Return values:***VOS_NO_ERR* no error*VOS_PARAM_ERR* parameter must not be NULL**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:

\leftrightarrow ***pTime*** Pointer to time value
 \leftarrow ***pCmp*** Pointer to time value to compare

Return values:

0 $pTime == pCmp$
-1 $pTime < pCmp$
1 $pTime > pCmp$

5.34.2.4 EXT_DECL VOS_ERR_T vos_divTime (VOS_TIME_T * *pTime*, UINT32 *div*)

Divide the first time by the second, return quotient in first.

Parameters:

\leftrightarrow ***pTime*** Pointer to time value
 \leftarrow ***div*** Divisor

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR parameter must not be NULL

Divide the first time by the second, return quotient in first.

Parameters:

\leftrightarrow ***pTime*** Pointer to time value
 \leftarrow ***div*** Divisor

Return values:

VOS_NO_ERR no error
VOS_PARAM_ERR parameter must not be NULL

5.34.2.5 EXT_DECL VOS_ERR_T vos_getTime (VOS_TIME_T * *pTime*)

Return the current time in sec and us.

Parameters:

\rightarrow ***pTime*** Pointer to time value

Return values:*VOS_NO_ERR* no error*VOS_INIT_ERR* module not initialised**Parameters:**→ *pTime* Pointer to time value**Return values:***VOS_NO_ERR* no error*VOS_PARAM_ERR* parameter out of range/invalid**5.34.2.6 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.

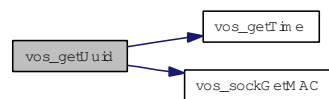
Return values:*timestamp* "yyyymmdd-hh:mm:ss.ms"**5.34.2.7 EXT_DECL VOS_ERR_T vos_getUuid (VOS_UUID_T pUuid)**

Get a universal unique identifier according to RFC 4122 time based version.

Parameters:→ *pUuid* Pointer to a universal unique identifier**Return values:***VOS_NO_ERR* no error*VOS_UNKNOWN_ERR* Could not create UUID**Parameters:**→ *pUuid* Pointer to a universal unique identifier

Return values:*VOS_NO_ERR* no error*VOS_UNKNOWN_ERR* Could not create UUID**Parameters:**→ *pUUID* Pointer to a universal unique identifier**Return values:***VOS_NO_ERR* no error*VOS_INIT_ERR* module not initialised

Here is the call graph for this function:

**5.34.2.8 EXT_DECL VOS_ERR_T 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**Return values:***VOS_NO_ERR* no error*VOS_PARAM_ERR* parameter must not be NULL**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:

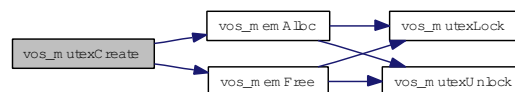
VOS_NO_ERR no error

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 VOS_ERR_T 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

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

VOS_MUTEX_ERR no such mutex

Release the resources taken by the mutex.

Parameters:

← *pMutex* mutex handle

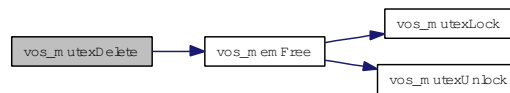
Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR pMutex == NULL or wrong type

VOS_MUTEX_ERR no such mutex

Here is the call graph for this function:



5.34.2.11 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

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR module not initialised

VOS_NOINIT_ERR invalid handle

Unlock the mutex.

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

Unlock the mutex.

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

5.34.2.14 EXT_DECL VOS_ERR_T vos_subTime (VOS_TIME_T * *pTime*, const VOS_TIME_T * *pSub*)

Subtract the second from the first time stamp, return diff in first.

Parameters:

↔ *pTime* Pointer to time value

← *pSub* Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

Parameters:

↔ *pTime* Pointer to time value

← *pSub* Pointer to time value

Return values:

VOS_NO_ERR no error

VOS_PARAM_ERR parameter must not be NULL

5.34.2.15 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:



5.34.2.16 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

5.34.2.17 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

5.34.2.18 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

5.34.2.19 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


```
VOS_THREAD_ERR = -13,  
VOS_UNKNOWN_ERR = -99 }  
Return codes for all VOS API functions.
```

- enum `VOS_LOG_T` {
 `VOS_LOG_ERROR` = 0,
 `VOS_LOG_WARNING` = 1,
 `VOS_LOG_INFO` = 2,
 `VOS_LOG_DBG` = 3 }
Categories for logging.

Functions

- EXT_DECL `VOS_ERR_T vos_init` (void *pRefCon, `VOS_PRINT_DBG_T` pDebugOutput)
Initialize the vos library.

5.35.1 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](#) 148 2012-11-19 12:46:34Z bloehr

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_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

Here is the call graph for this function:



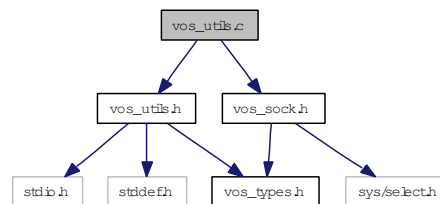
5.36 vos_utils.c File Reference

Common functions for VOS.

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

```
#include "vos_sock.h"
```

Include dependency graph for vos_utils.c:



Functions

- **VOS_ERR_T vos_init** (void *pRefCon, **VOS_PRINT_DBG_T** pDebugOutput)
Initialize the vos library.
- **UINT32 vos_crc32** (UINT32 crc, const **UINT8** *pData, **UINT32** dataLen)
Compute crc32 according to IEEE802.3.

5.36.1 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](#) 150 2012-11-19 12:53:43Z bloehr

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 vos library.

This is used to set the output function for all VOS error and debug output.

Parameters:

- ← **pRefCon* user context
- ← **pDebugOutput* pointer to debug output function

Return values:

VOS_NO_ERR no error

VOS_INIT_ERR unsupported

Here is the call graph for this function:

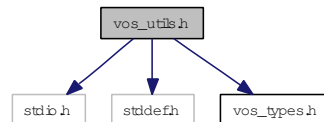


5.37 vos_utils.h File Reference

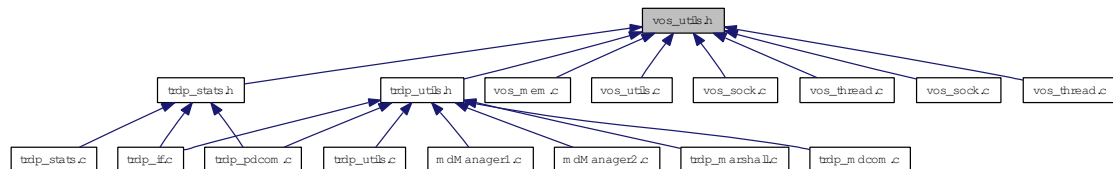
Typedefs for OS abstraction.

```
#include <stdio.h>
#include <stddef.h>
#include "vos_types.h"
```

Include dependency graph for vos_utils.h:



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



Defines

- #define [VOS_MAX_PRNT_STR_SIZE](#) 256
String size definitions for the debug output functions.
- #define [VOS_MAX_FRMT_SIZE](#) 64
Max.
- #define [VOS_MAX_ERR_STR_SIZE](#) (VOS_MAX_PRNT_STR_SIZE - VOS_MAX_FRMT_SIZE)
Max.
- #define [vos_print](#)(level, string)
Debug output macro without formatting options.
- #define [vos_printf](#)(level, format, args...)
Debug output macro with formatting options.
- #define [ALIGNOF](#)(type) offsetof(struct { char c; type member; }, member)
Alignment macros.

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](#) 149 2012-11-19 12:51:57Z bloehr

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, [226](#)
 - trdp_utils.h, [233](#)
- cycleTime
 - TRDP_PROCESS_CONFIG_T, [44](#)
- cyclicThread
 - posix/vos_thread.c, [302](#)
 - windows/vos_thread.c, [314](#)
- datasetLength
 - GNU_PACKED, [10](#)
- dbgOut
 - echoPolling.c, [62](#)
 - echoSelect.c, [66](#)
- destAddr
 - TRDP_PUB_STATISTICS_T, [47](#)
- echoPolling.c, [61](#)
 - dbgOut, [62](#)
 - main, [62](#)
- echoSelect.c, [65](#)
 - dbgOut, [66](#)
 - main, [66](#)
 - myPDcallBack, [69](#)
- filterAddr
 - TRDP_SUBS_STATISTICS_T, [55](#)
- GNU_PACKED, [9](#)
 - datasetLength, [10](#)
 - msgType, [10](#)
 - protocolVersion, [10](#)
- leaderName
 - TRDP_PROCESS_CONFIG_T, [44](#)
- main
 - echoPolling.c, [62](#)
 - echoSelect.c, [66](#)
 - sendHello.c, [75](#)
- MD_ELE, [12](#)
 - pktFlags, [13](#)
- mdManager1.c, [70](#)
- mdManager2.c, [72](#)
- msgType
 - GNU_PACKED, [10](#)
 - TRDP_MD_INFO_T, [34](#)
 - TRDP_PD_INFO_T, [41](#)
- myPDcallBack
 - echoSelect.c, [69](#)
- numRecv
 - TRDP_SUBS_STATISTICS_T, [56](#)
- operator
 - TRDP_TRAIN_INFO_T, [58](#)
- options
 - TRDP_PROCESS_CONFIG_T, [45](#)
- orient
 - TRDP_CAR_INFO_T, [19](#)
 - TRDP_CST_INFO_T, [22](#)
 - TRDP_DEVICE_INFO_T, [27](#)
- owner
 - TRDP_CST_INFO_T, [22](#)
- pCarInfo
 - TRDP_CST_INFO_T, [22](#)
- pCstInfo
 - TRDP_TRAIN_INFO_T, [58](#)
- PD_ELE, [15](#)
- pDevInfo
 - TRDP_CAR_INFO_T, [19](#)
- pFctInfo
 - TRDP_CST_INFO_T, [22](#)
- pktFlags
 - MD_ELE, [13](#)
- posix/vos_private.h
 - vos_mutexLocalCreate, [251](#)
 - vos_mutexLocalDelete, [251](#)
- posix/vos_sock.c
 - vos_dottedIP, [256](#)
 - vos_htonl, [257](#)
 - vos_htons, [257](#)
 - vos_ipDotted, [257](#)
 - vos_isMulticast, [257](#)
 - vos_ntohl, [257](#)
 - vos_ntohs, [258](#)
 - vos_sockAccept, [258](#)
 - vos_sockBind, [258](#)
 - vos_sockClose, [259](#)

- vos_sockConnect, 259
- vos_sockGetMAC, 260
- vos_sockInit, 260
- vos_sockJoinMC, 260
- vos_sockLeaveMC, 261
- vos_sockListen, 261
- vos_sockOpenTCP, 262
- vos_sockOpenUDP, 262
- vos_sockReceiveTCP, 262
- vos_sockReceiveUDP, 263
- vos_sockSendTCP, 263
- vos_sockSendUDP, 264
- vos_sockSetOptions, 264
- posix/vos_thread.c
 - cyclicThread, 302
 - vos_addTime, 303
 - vos_clearTime, 303
 - vos_cmpTime, 303
 - vos_divTime, 303
 - vos_getTime, 304
 - vos_getTimeStamp, 304
 - vos_getUuid, 304
 - vos_mulTime, 305
 - vos_mutexCreate, 305
 - vos_mutexDelete, 305
 - vos_mutexLocalCreate, 306
 - vos_mutexLocalDelete, 306
 - vos_mutexLock, 306
 - vos_mutexTryLock, 307
 - vos_mutexUnlock, 307
 - vos_semaCreate, 307
 - vos_semaDelete, 308
 - vos_semaGive, 308
 - vos_semaTake, 308
 - vos_subTime, 309
 - vos_threadCreate, 309
 - vos_threadDelay, 310
 - vos_threadInit, 310
 - vos_threadIsActive, 310
 - vos_threadTerminate, 310
- priority
 - TRDP_PROCESS_CONFIG_T, 44
- protocolVersion
 - GNU_PACKED, 10
- qos
 - VOS_SOCK_OPT_T, 59
- sendHello.c, 74
 - main, 75
- tau_tci.h
 - TRDP_FCT_CAR, 94
 - TRDP_FCT_CST, 94
 - TRDP_FCT_INVALID, 94
 - TRDP_FCT_TRAIN, 94
 - 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, 103
 - TRDP_DBG_DBG, 103
 - TRDP_DBG_DEFAULT, 102
 - TRDP_DBG_ERR, 103
 - TRDP_DBG_INFO, 103
 - TRDP_DBG_LOC, 103
 - TRDP_DBG_OFF, 102
 - TRDP_DBG_TIME, 103
 - TRDP_DBG_WARN, 103
- tau_addr.h, 77
 - tau_addr2CarId, 79
 - tau_addr2CarNo, 79
 - tau_addr2CstId, 80
 - tau_addr2CstNo, 80
 - tau_addr2IecCarNo, 80
 - tau_addr2IecCstNo, 81
 - tau_addr2Uri, 81
 - tau_carNo2Ids, 81
 - tau_cstNo2CstId, 82
 - tau_getOwnAddr, 82
 - tau_getOwnIds, 82
 - tau_iecCarNo2Ids, 83
 - tau_iecCstNo2CstId, 83
 - tau_label2CarId, 83
 - tau_label2CarNo, 84
 - tau_label2CstId, 84
 - tau_label2CstNo, 84
 - tau_label2IecCarNo, 85
 - tau_label2IecCstNo, 85
 - tau_uri2Addr, 85
- tau_addr2CarId
 - tau_addr.h, 79
- tau_addr2CarNo
 - tau_addr.h, 79
- tau_addr2CstId
 - tau_addr.h, 80
- tau_addr2CstNo
 - tau_addr.h, 80
- tau_addr2IecCarNo
 - tau_addr.h, 80
- tau_addr2IecCstNo
 - tau_addr.h, 81
- tau_addr2Uri
 - tau_addr.h, 81
- tau_calcDatasetSize
 - tau_marshall.h, 88
- tau_carNo2Ids

- tau_addr.h, 81
- tau_cstNo2CstId
 - tau_addr.h, 82
- tau_getCarDevCnt
 - tau_tci.h, 95
- tau_getCarInfo
 - tau_tci.h, 95
- tau_getCarOrient
 - tau_tci.h, 96
- tau_getCstCarCnt
 - tau_tci.h, 96
- tau_getCstFctCnt
 - tau_tci.h, 96
- tau_getCstFctInfo
 - tau_tci.h, 97
- tau_getCstInfo
 - tau_tci.h, 97
- tau_getDevInfo
 - tau_tci.h, 97
- tau_getEthState
 - tau_tci.h, 98
- tau_getIecCarOrient
 - tau_tci.h, 98
- tau_getOwnAddr
 - tau_addr.h, 82
- tau_getOwnIds
 - tau_addr.h, 82
- tau_getTrnCarCnt
 - tau_tci.h, 99
- tau_getTrnCstCnt
 - tau_tci.h, 99
- tau_getTrnInfo
 - tau_tci.h, 99
- tau_iecCarNo2Ids
 - tau_addr.h, 83
- tau_iecCstNo2CstId
 - tau_addr.h, 83
- tau_initMarshall
 - tau_marshall.h, 89
- tau_label2CarId
 - tau_addr.h, 83
- tau_label2CarNo
 - tau_addr.h, 84
- tau_label2CstId
 - tau_addr.h, 84
- tau_label2CstNo
 - tau_addr.h, 84
- tau_label2IecCarNo
 - tau_addr.h, 85
- tau_label2IecCstNo
 - tau_addr.h, 85
- tau_marshall
 - tau_marshall.h, 90
- tau_marshall.h, 87
 - tau_calcDatasetSize, 88
 - tau_initMarshall, 89
 - tau_marshall, 90
 - tau_marshallDs, 88
 - tau_unmarshall, 90
 - tau_unmarshallDs, 89
- TAU_MARSHALL_INFO_T, 17
- tau_marshallDs
 - tau_marshall.h, 88
- tau_readXmlConfig
 - tau_xml.h, 103
- tau_readXmlDatasetConfig
 - tau_xml.h, 103
- tau_tci.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_getEthState, 98
 - tau_getIecCarOrient, 98
 - tau_getTrnCarCnt, 99
 - tau_getTrnCstCnt, 99
 - tau_getTrnInfo, 99
- TRDP_FCT_T, 94
- TRDP_INAUG_STATE_T, 94
- tau_types.h, 100
- tau_unmarshall
 - tau_marshall.h, 90
- tau_unmarshallDs
 - tau_marshall.h, 89
- tau_uri2Addr
 - tau_addr.h, 85
- tau_xml.h, 101
 - tau_readXmlConfig, 103
 - tau_readXmlDatasetConfig, 103
 - TRDP_DBG_OPTION_T, 102
- timeout
 - TRDP_SUBS_STATISTICS_T, 55
- tlc_closeSession
 - trdp_if.c, 108
 - trdp_if_light.h, 138
- tlc_freeBuf
 - trdp_if_light.h, 139
- tlc_getInterval
 - trdp_if.c, 109
 - trdp_if_light.h, 139
- tlc_getJoinStatistics
 - trdp_if_light.h, 140
 - trdp_stats.c, 205
- tlc_getListStatistics

- trdp_if_light.h, 141
- trdp_stats.c, 206
- tlc_getPubStatistics
 - trdp_if_light.h, 142
 - trdp_stats.c, 206
- tlc_getRedStatistics
 - trdp_if_light.h, 142
 - trdp_stats.c, 207
- tlc_getStatistics
 - trdp_if_light.h, 143
 - trdp_stats.c, 207
- tlc_getSubsStatistics
 - trdp_if_light.h, 144
 - trdp_stats.c, 208
- tlc_getVersion
 - trdp_if.c, 109
 - trdp_if_light.h, 145
- tlc_init
 - trdp_if.c, 110
 - trdp_if_light.h, 145
- tlc_openSession
 - trdp_if.c, 110
 - trdp_if_light.h, 146
- tlc_process
 - trdp_if.c, 111
 - trdp_if_light.h, 147
- tlc_reinitSession
 - trdp_if.c, 114
 - trdp_if_light.h, 150
- tlc_resetStatistics
 - trdp_if_light.h, 150
 - trdp_stats.c, 208
- tlc_setTopoCount
 - trdp_if.c, 114
 - trdp_if_light.h, 151
- tlc_terminate
 - trdp_if.c, 114
 - trdp_if_light.h, 151
- tlm_abortSession
 - trdp_if_light.h, 152
- tlm_addListener
 - trdp_if.c, 115
 - trdp_if_light.h, 152
- tlm_confirm
 - trdp_if.c, 116
 - trdp_if_light.h, 154
- tlm_delListener
 - trdp_if.c, 116
 - trdp_if_light.h, 155
- tlm_notify
 - trdp_if.c, 117
 - trdp_if_light.h, 156
- tlm_reply
 - trdp_if.c, 118
 - trdp_if_light.h, 157
- tlm_replyErr
 - trdp_if.c, 118
 - trdp_if_light.h, 158
- tlm_replyQuery
 - trdp_if.c, 119
 - trdp_if_light.h, 160
- tlm_request
 - trdp_if.c, 120
 - trdp_if_light.h, 161
- tlp_get
 - trdp_if.c, 121
 - trdp_if_light.h, 162
- tlp_getRedundant
 - trdp_if.c, 122
 - trdp_if_light.h, 164
- tlp_publish
 - trdp_if.c, 123
 - trdp_if_light.h, 165
- tlp_put
 - trdp_if.c, 124
 - trdp_if_light.h, 167
- tlp_request
 - trdp_if.c, 125
 - trdp_if_light.h, 168
- tlp_setRedundant
 - trdp_if.c, 126
 - trdp_if_light.h, 170
- tlp_subscribe
 - trdp_if.c, 127
 - trdp_if_light.h, 171
- tlp_unpublish
 - trdp_if.c, 128
 - trdp_if_light.h, 173
- tlp_unsubscribe
 - trdp_if.c, 129
 - trdp_if_light.h, 174
- toBehav
 - TRDP_SUBS_STATISTICS_T, 55
- topoCnt
 - TRDP_TRAIN_INFO_T, 58
- TRDP_APPTIMEOUT_ERR
 - trdp_types.h, 222
- TRDP_BOOLEAN
 - trdp_types.h, 221
- TRDP_CHAR8
 - trdp_types.h, 221
- TRDP_COMID_ERR
 - trdp_types.h, 222
- TRDP_CRC_ERR
 - trdp_types.h, 222
- TRDP_DBG_CAT
 - tau_xml.h, 103
- TRDP_DBG_DBG

- [tau_xml.h](#), 103
- TRDP_DBG_DEFAULT
 - [tau_xml.h](#), 102
- TRDP_DBG_ERR
 - [tau_xml.h](#), 103
- TRDP_DBG_INFO
 - [tau_xml.h](#), 103
- TRDP_DBG_LOC
 - [tau_xml.h](#), 103
- TRDP_DBG_OFF
 - [tau_xml.h](#), 102
- TRDP_DBG_TIME
 - [tau_xml.h](#), 103
- TRDP_DBG_WARN
 - [tau_xml.h](#), 103
- TRDP_FCT_CAR
 - [tau_tci.h](#), 94
- TRDP_FCT_CST
 - [tau_tci.h](#), 94
- TRDP_FCT_INVALID
 - [tau_tci.h](#), 94
- TRDP_FCT_TRAIN
 - [tau_tci.h](#), 94
- TRDP_FLAGS_CALLBACK
 - [trdp_types.h](#), 222
- TRDP_FLAGS_MARSHALL
 - [trdp_types.h](#), 222
- TRDP_FLAGS_REDUNDANT
 - [trdp_types.h](#), 222
- TRDP_FLAGS_TCP
 - [trdp_types.h](#), 222
- TRDP_INAUG_INVALID
 - [tau_tci.h](#), 95
- TRDP_INAUG_LEAD_CONF
 - [tau_tci.h](#), 95
- TRDP_INAUG_LEAD_UNCONF
 - [tau_tci.h](#), 95
- TRDP_INAUG_NOLEAD_UNCONF
 - [tau_tci.h](#), 95
- TRDP_INIT_ERR
 - [trdp_types.h](#), 221
- TRDP_INT16
 - [trdp_types.h](#), 221
- TRDP_INT32
 - [trdp_types.h](#), 221
- TRDP_INT64
 - [trdp_types.h](#), 221
- TRDP_INT8
 - [trdp_types.h](#), 221
- TRDP_IO_ERR
 - [trdp_types.h](#), 222
- TRDP_MD_ELE_ST_NONE
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_RX_ARM
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_RX_REPLY_W4AP_CONF
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_RX_REQ_W4AP_REPLY
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_TX_CONFIRM_ARM
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_TX_ERROR_ARM
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_TX_NOTIFY_ARM
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_TX_REPLY_ARM
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_TX_REPLYQUERY_ARM
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_TX_REPLYQUERY_W4C
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_TX_REQUEST_ARM
 - [trdp_private.h](#), 202
- TRDP_MD_ELE_ST_TX_REQUEST_W4Y
 - [trdp_private.h](#), 202
- TRDP_MEM_ERR
 - [trdp_types.h](#), 222
- TRDP_MSG_MC
 - [trdp_types.h](#), 222
- TRDP_MSG_ME
 - [trdp_types.h](#), 222
- TRDP_MSG_MN
 - [trdp_types.h](#), 222
- TRDP_MSG_MP
 - [trdp_types.h](#), 222
- TRDP_MSG_MQ
 - [trdp_types.h](#), 222
- TRDP_MSG_MR
 - [trdp_types.h](#), 222
- TRDP_MSG_PD
 - [trdp_types.h](#), 222
- TRDP_MSG_PE
 - [trdp_types.h](#), 222
- TRDP_MSG_PR
 - [trdp_types.h](#), 222
- TRDP_MUTEX_ERR
 - [trdp_types.h](#), 222
- TRDP_NO_ERR
 - [trdp_types.h](#), 221
- TRDP_NODATA_ERR
 - [trdp_types.h](#), 221
- TRDP_NOINIT_ERR
 - [trdp_types.h](#), 221
- TRDP_NOLIST_ERR
 - [trdp_types.h](#), 222
- TRDP_NOPUB_ERR
 - [trdp_types.h](#), 222
- TRDP_NOSESSION_ERR

- trdp_types.h, [222](#)
- TRDP_NOSUB_ERR
 - trdp_types.h, [222](#)
- TRDP_OPTION_BLOCK
 - trdp_types.h, [223](#)
- TRDP_OPTION_TRAFFIC_SHAPING
 - trdp_types.h, [223](#)
- TRDP_PARAM_ERR
 - trdp_types.h, [221](#)
- trdp_private.h
 - TRDP_MD_ELE_ST_NONE, [202](#)
 - TRDP_MD_ELE_ST_RX_ARM, [202](#)
 - TRDP_MD_ELE_ST_RX_REPLY_W4AP_-
CONF, [202](#)
 - TRDP_MD_ELE_ST_RX_REQ_W4AP_-
REPLY, [202](#)
 - TRDP_MD_ELE_ST_TX_CONFIRM_ARM,
[202](#)
 - TRDP_MD_ELE_ST_TX_ERROR_ARM,
[202](#)
 - TRDP_MD_ELE_ST_TX_NOTIFY_ARM,
[202](#)
 - TRDP_MD_ELE_ST_TX_REPLY_ARM,
[202](#)
 - TRDP_MD_ELE_ST_TX_REPLYQUERY_-
ARM, [202](#)
 - TRDP_MD_ELE_ST_TX_REPLYQUERY_-
W4C, [202](#)
 - TRDP_MD_ELE_ST_TX_REQUEST_ARM,
[202](#)
 - TRDP_MD_ELE_ST_TX_REQUEST_W4Y,
[202](#)
 - TRDP_PULL_SUB, [202](#)
 - TRDP_REQ_2B_SENT, [202](#)
 - TRDP SOCK_MD_TCP, [203](#)
 - TRDP SOCK_MD_UDP, [203](#)
 - TRDP SOCK_PD, [203](#)
 - TRDP_TIMED_OUT, [202](#)
- TRDP_PULL_SUB
 - trdp_private.h, [202](#)
- TRDP_QUEUE_ERR
 - trdp_types.h, [222](#)
- TRDP_QUEUE_FULL_ERR
 - trdp_types.h, [222](#)
- TRDP_REAL32
 - trdp_types.h, [221](#)
- TRDP_REAL64
 - trdp_types.h, [221](#)
- TRDP_RED_FOLLOWER
 - trdp_types.h, [223](#)
- TRDP_RED_LEADER
 - trdp_types.h, [223](#)
- TRDP_REQ_2B_SENT
 - trdp_private.h, [202](#)
- TRDP_SEMA_ERR
 - trdp_types.h, [222](#)
- TRDP_SESSION_ABORT_ERR
 - trdp_types.h, [222](#)
- TRDP SOCK_ERR
 - trdp_types.h, [222](#)
- TRDP SOCK_MD_TCP
 - trdp_private.h, [203](#)
- TRDP SOCK_MD_UDP
 - trdp_private.h, [203](#)
- TRDP SOCK_PD
 - trdp_private.h, [203](#)
- TRDP_STATE_ERR
 - trdp_types.h, [222](#)
- TRDP_TIMED_OUT
 - trdp_private.h, [202](#)
- TRDP_TIMEDATE32
 - trdp_types.h, [221](#)
- TRDP_TIMEDATE48
 - trdp_types.h, [221](#)
- TRDP_TIMEDATE64
 - trdp_types.h, [221](#)
- TRDP_TIMEOUT_ERR
 - trdp_types.h, [221](#)
- TRDP_TO_KEEP_LAST_VALUE
 - trdp_types.h, [223](#)
- TRDP_TO_SET_TO_ZERO
 - trdp_types.h, [223](#)
- TRDP_TOPO_ERR
 - trdp_types.h, [222](#)
- TRDP_TYPE_MAX
 - trdp_types.h, [221](#)
- trdp_types.h
 - TRDP_APPTIMEOUT_ERR, [222](#)
 - TRDP_BOOLEAN, [221](#)
 - TRDP_CHAR8, [221](#)
 - TRDP_COMID_ERR, [222](#)
 - TRDP_CRC_ERR, [222](#)
 - TRDP_FLAGS_CALLBACK, [222](#)
 - TRDP_FLAGS_MARSHALL, [222](#)
 - TRDP_FLAGS_REDUNDANT, [222](#)
 - TRDP_FLAGS_TCP, [222](#)
 - TRDP_INIT_ERR, [221](#)
 - TRDP_INT16, [221](#)
 - TRDP_INT32, [221](#)
 - TRDP_INT64, [221](#)
 - TRDP_INT8, [221](#)
 - TRDP_IO_ERR, [222](#)
 - TRDP_MEM_ERR, [222](#)
 - TRDP_MSG_MC, [222](#)
 - TRDP_MSG_ME, [222](#)
 - TRDP_MSG_MN, [222](#)
 - TRDP_MSG_MP, [222](#)
 - TRDP_MSG_MQ, [222](#)

- TRDP_MSG_MR, [222](#)
- TRDP_MSG_PD, [222](#)
- TRDP_MSG_PE, [222](#)
- TRDP_MSG_PR, [222](#)
- TRDP_MUTEX_ERR, [222](#)
- TRDP_NO_ERR, [221](#)
- TRDP_NODATA_ERR, [221](#)
- TRDP_NOINIT_ERR, [221](#)
- TRDP_NOLIST_ERR, [222](#)
- TRDP_NOPUB_ERR, [222](#)
- TRDP_NOSESSION_ERR, [222](#)
- TRDP_NOSUB_ERR, [222](#)
- TRDP_OPTION_BLOCK, [223](#)
- TRDP_OPTION_TRAFFIC_SHAPING, [223](#)
- TRDP_PARAM_ERR, [221](#)
- TRDP_QUEUE_ERR, [222](#)
- TRDP_QUEUE_FULL_ERR, [222](#)
- TRDP_REAL32, [221](#)
- TRDP_REAL64, [221](#)
- TRDP_RED_FOLLOWER, [223](#)
- TRDP_RED_LEADER, [223](#)
- TRDP_SEMA_ERR, [222](#)
- TRDP_SESSION_ABORT_ERR, [222](#)
- TRDP_SOCKET_ERR, [222](#)
- TRDP_STATE_ERR, [222](#)
- TRDP_TIMEDATE32, [221](#)
- TRDP_TIMEDATE48, [221](#)
- TRDP_TIMEDATE64, [221](#)
- TRDP_TIMEOUT_ERR, [221](#)
- TRDP_TO_KEEP_LAST_VALUE, [223](#)
- TRDP_TO_SET_TO_ZERO, [223](#)
- TRDP_TOPO_ERR, [222](#)
- TRDP_TYPE_MAX, [221](#)
- TRDP_UINT16, [221](#)
- TRDP_UINT32, [221](#)
- TRDP_UINT64, [221](#)
- TRDP_UINT8, [221](#)
- TRDP_UNKNOWN_ERR, [222](#)
- TRDP_UTF16, [221](#)
- TRDP_UINT16
 - trdp_types.h, [221](#)
- TRDP_UINT32
 - trdp_types.h, [221](#)
- TRDP_UINT64
 - trdp_types.h, [221](#)
- TRDP_UINT8
 - trdp_types.h, [221](#)
- TRDP_UNKNOWN_ERR
 - trdp_types.h, [222](#)
- TRDP_UTF16
 - trdp_types.h, [221](#)
- TRDP_CAR_INFO_T, [18](#)
 - orient, [19](#)
 - pDevInfo, [19](#)
- TRDP_COMID_DSID_MAP_T, [20](#)
- TRDP_COMID_ECHO
 - trdp_types.h, [218](#)
- TRDP_CST_INFO_T, [21](#)
 - orient, [22](#)
 - owner, [22](#)
 - pCarInfo, [22](#)
 - pFctInfo, [22](#)
- TRDP_DATA_TYPE_T
 - trdp_types.h, [221](#)
- TRDP_DATASET_ELEMENT_T, [23](#)
 - type, [23](#)
- TRDP_DATASET_T, [24](#)
- TRDP_DBG_CONFIG_T, [25](#)
- TRDP_DBG_OPTION_T
 - tau_xml.h, [102](#)
- TRDP_DEVICE_INFO_T, [26](#)
 - orient, [27](#)
- TRDP_ERR_T
 - trdp_types.h, [221](#)
- TRDP_FCT_INFO_T, [28](#)
- TRDP_FCT_T
 - tau_tci.h, [94](#)
- TRDP_FLAGS_T
 - trdp_types.h, [222](#)
- trdp_getSeqCnt
 - trdp_utils.c, [226](#)
 - trdp_utils.h, [233](#)
- trdp_getTopoCount
 - trdp_if.c, [129](#)
 - trdp_if.h, [132](#)
- TRDP_HANDLE, [29](#)
 - trdp_if.c, [105](#)
 - tlc_closeSession, [108](#)
 - tlc_getInterval, [109](#)
 - tlc_getVersion, [109](#)
 - tlc_init, [110](#)
 - tlc_openSession, [110](#)
 - tlc_process, [111](#)
 - tlc_reinitSession, [114](#)
 - tlc_setTopoCount, [114](#)
 - tlc_terminate, [114](#)
 - tlm_addListener, [115](#)
 - tlm_confirm, [116](#)
 - tlm_delListener, [116](#)
 - tlm_notify, [117](#)
 - tlm_reply, [118](#)
 - tlm_replyErr, [118](#)
 - tlm_replyQuery, [119](#)
 - tlm_request, [120](#)
 - tlp_get, [121](#)
 - tlp_getRedundant, [122](#)
 - tlp_publish, [123](#)
 - tlp_put, [124](#)

- tlp_request, 125
- tlp_setRedundant, 126
- tlp_subscribe, 127
- tlp_unpublish, 128
- tlp_unsubscribe, 129
- trdp_getTopoCount, 129
- trdp_isValidSession, 130
- trdp_sessionQueue, 130
- trdp_if.h, 131
 - trdp_getTopoCount, 132
 - trdp_isValidSession, 132
 - trdp_sessionQueue, 132
- trdp_if_light.h, 134
 - tlc_closeSession, 138
 - tlc_freeBuf, 139
 - tlc_getInterval, 139
 - tlc_getJoinStatistics, 140
 - tlc_getListStatistics, 141
 - tlc_getPubStatistics, 142
 - tlc_getRedStatistics, 142
 - tlc_getStatistics, 143
 - tlc_getSubsStatistics, 144
 - tlc_getVersion, 145
 - tlc_init, 145
 - tlc_openSession, 146
 - tlc_process, 147
 - tlc_reinitSession, 150
 - tlc_resetStatistics, 150
 - tlc_setTopoCount, 151
 - tlc_terminate, 151
 - tlm_abortSession, 152
 - tlm_addListener, 152
 - tlm_confirm, 154
 - tlm_delListener, 155
 - tlm_notify, 156
 - tlm_reply, 157
 - tlm_replyErr, 158
 - tlm_replyQuery, 160
 - tlm_request, 161
 - tlp_get, 162
 - tlp_getRedundant, 164
 - tlp_publish, 165
 - tlp_put, 167
 - tlp_request, 168
 - tlp_setRedundant, 170
 - tlp_subscribe, 171
 - tlp_unpublish, 173
 - tlp_unsubscribe, 174
- TRDP_INAUG_STATE_T
 - tau_tci.h, 94
- trdp_initSockets
 - trdp_utils.c, 226
 - trdp_utils.h, 233
- trdp_initStats
 - trdp_stats.c, 209
 - trdp_stats.h, 212
- TRDP_IP_ADDR_T
 - trdp_types.h, 219
- trdp_isRcvSeqCnt
 - trdp_utils.c, 226
 - trdp_utils.h, 233
- trdp_isValidSession
 - trdp_if.c, 130
 - trdp_if.h, 132
- TRDP_LIST_STATISTICS_T, 30
- trdp_marshall.c, 176
- TRDP_MARSHALL_CONFIG_T, 31
- TRDP_MARSHALL_T
 - trdp_types.h, 219
- TRDP_MAX_FILE_NAME_LEN
 - trdp_types.h, 218
- TRDP_MAX_LABEL_LEN
 - trdp_types.h, 218
- TRDP_MAX_URI_HOST_LEN
 - trdp_types.h, 218
- TRDP_MAX_URI_LEN
 - trdp_types.h, 219
- TRDP_MAX_URI_USER_LEN
 - trdp_types.h, 219
- TRDP_MD_CALLBACK_T
 - trdp_types.h, 219
- TRDP_MD_CONFIG_T, 32
- TRDP_MD_ELE_ST_T
 - trdp_private.h, 202
- TRDP_MD_INFO_T, 33
 - msgType, 34
- TRDP_MD_STATISTICS_T, 35
- trdp_mdCheck
 - trdp_mdcom.c, 179
- trdp_mdcom.c, 178
 - trdp_mdCheck, 179
 - trdp_mdReceive, 179
 - trdp_mdRecv, 180
 - trdp_mdSend, 181
 - trdp_mdUpdate, 181
 - trdp_rcvMD, 181
 - trdp_sendMD, 182
- trdp_mdcom.h, 183
 - trdp_mdReceive, 184
 - trdp_mdSend, 185
 - trdp_mdUpdate, 185
- trdp_MDqueueDelElement
 - trdp_utils.c, 227
 - trdp_utils.h, 234
- trdp_MDqueueFindAddr
 - trdp_utils.c, 227
 - trdp_utils.h, 234
- trdp_MDqueueInsFirst

- trdp_utils.c, 227
- trdp_utils.h, 234
- trdp_mdReceive
 - trdp_mdcom.c, 179
 - trdp_mdcom.h, 184
- trdp_mdRecv
 - trdp_mdcom.c, 180
- trdp_mdSend
 - trdp_mdcom.c, 181
 - trdp_mdcom.h, 185
- trdp_mdUpdate
 - trdp_mdcom.c, 181
 - trdp_mdcom.h, 185
- TRDP_MEM_CONFIG_T, 37
- TRDP_MEM_STATISTICS_T, 38
- TRDP_MSG_T
 - trdp_types.h, 222
- TRDP_OPTION_T
 - trdp_types.h, 222
- trdp_packetSizePD
 - trdp_utils.c, 227
 - trdp_utils.h, 235
- TRDP_PD_CALLBACK_T
 - trdp_types.h, 220
- TRDP_PD_CONFIG_T, 39
- TRDP_PD_INFO_T, 40
 - msgType, 41
- TRDP_PD_STATISTICS_T, 42
- trdp_pdCheck
 - trdp_pdcom.c, 188
 - trdp_pdcom.h, 194
- trdp_pdcom.c, 187
 - trdp_pdCheck, 188
 - trdp_pdDataUpdate, 189
 - trdp_pdDistribute, 189
 - trdp_pdInit, 189
 - trdp_pdReceive, 190
 - trdp_pdSend, 191
 - trdp_pdSendQueued, 192
 - trdp_pdUpdate, 192
- trdp_pdcom.h, 193
 - trdp_pdCheck, 194
 - trdp_pdDataUpdate, 195
 - trdp_pdDistribute, 195
 - trdp_pdInit, 195
 - trdp_pdReceive, 196
 - trdp_pdSend, 197
 - trdp_pdSendQueued, 198
 - trdp_pdUpdate, 198
- trdp_pdDataUpdate
 - trdp_pdcom.c, 189
 - trdp_pdcom.h, 195
- trdp_pdDistribute
 - trdp_pdcom.c, 189
 - trdp_pdcom.h, 195
- trdp_pdInit
 - trdp_pdcom.c, 189
 - trdp_pdcom.h, 195
- trdp_pdPrepareStats
 - trdp_stats.c, 209
 - trdp_stats.h, 212
- trdp_pdReceive
 - trdp_pdcom.c, 190
 - trdp_pdcom.h, 196
- trdp_pdSend
 - trdp_pdcom.c, 191
 - trdp_pdcom.h, 197
- trdp_pdSendQueued
 - trdp_pdcom.c, 192
 - trdp_pdcom.h, 198
- trdp_pdUpdate
 - trdp_pdcom.c, 192
 - trdp_pdcom.h, 198
- TRDP_PRINT_DBG_T
 - trdp_types.h, 220
- TRDP_PRIV_FLAGS_T
 - trdp_private.h, 202
- trdp_private.h, 199
 - TRDP_MD_ELE_ST_T, 202
 - TRDP_PRIV_FLAGS_T, 202
 - TRDP SOCK_TYPE_T, 202
- TRDP_PROCESS_CONFIG_T, 44
 - cycleTime, 44
 - leaderName, 44
 - options, 45
 - priority, 44
- TRDP_PROP_INFO_T, 46
- TRDP_PUB_STATISTICS_T, 47
 - destAddr, 47
- trdp_queueAppLast
 - trdp_utils.c, 228
 - trdp_utils.h, 235
- trdp_queueDelElement
 - trdp_utils.c, 228
 - trdp_utils.h, 235
- trdp_queueFindComId
 - trdp_utils.c, 228
 - trdp_utils.h, 235
- trdp_queueFindPubAddr
 - trdp_utils.c, 228
 - trdp_utils.h, 235
- trdp_queueFindSubAddr
 - trdp_utils.c, 229
 - trdp_utils.h, 236
- trdp_queueInsFirst
 - trdp_utils.c, 229
 - trdp_utils.h, 236
- trdp_rcvMD

- trdp_mdcom.c, 181
- TRDP_RED_STATE_T
 - trdp_types.h, 223
- TRDP_RED_STATISTICS_T, 48
- trdp_releaseSocket
 - trdp_utils.c, 229
 - trdp_utils.h, 236
- trdp_requestSocket
 - trdp_utils.c, 229
 - trdp_utils.h, 237
- TRDP_SEND_PARAM_T, 49
- trdp_sendMD
 - trdp_mdcom.c, 182
- TRDP_SESSION, 50
- trdp_sessionQueue
 - trdp_if.c, 130
 - trdp_if.h, 132
- TRDP SOCK_TYPE_T
 - trdp_private.h, 202
- TRDP_SOCKETS, 52
 - usage, 52
- TRDP_STATISTICS_REQUEST_DSID
 - trdp_types.h, 219
- TRDP_STATISTICS_T, 53
- trdp_stats.c, 204
 - tlc_getJoinStatistics, 205
 - tlc_getListStatistics, 206
 - tlc_getPubStatistics, 206
 - tlc_getRedStatistics, 207
 - tlc_getStatistics, 207
 - tlc_getSubsStatistics, 208
 - tlc_resetStatistics, 208
 - trdp_initStats, 209
 - trdp_pdPrepareStats, 209
 - trdp_UpdateStats, 210
- trdp_stats.h, 211
 - trdp_initStats, 212
 - trdp_pdPrepareStats, 212
- TRDP_SUBS_STATISTICS_T, 55
 - filterAddr, 55
 - numRecv, 56
 - timeout, 55
 - toBehav, 55
- TRDP_TIME_T
 - trdp_types.h, 220
- TRDP_TO_BEHAVIOR_T
 - trdp_types.h, 223
- TRDP_TRAIN_INFO_T, 57
 - operator, 58
 - pCstInfo, 58
 - topoCnt, 58
- trdp_types.h, 213
 - TRDP_COMID_ECHO, 218
 - TRDP_DATA_TYPE_T, 221
 - TRDP_ERR_T, 221
 - TRDP_FLAGS_T, 222
 - TRDP_IP_ADDR_T, 219
 - TRDP_MARSHALL_T, 219
 - TRDP_MAX_FILE_NAME_LEN, 218
 - TRDP_MAX_LABEL_LEN, 218
 - TRDP_MAX_URI_HOST_LEN, 218
 - TRDP_MAX_URI_LEN, 219
 - TRDP_MAX_URI_USER_LEN, 219
 - TRDP_MD_CALLBACK_T, 219
 - TRDP_MSG_T, 222
 - TRDP_OPTION_T, 222
 - TRDP_PD_CALLBACK_T, 220
 - TRDP_PRINT_DBG_T, 220
 - TRDP_RED_STATE_T, 223
 - TRDP_STATISTICS_REQUEST_DSID, 219
 - TRDP_TIME_T, 220
 - TRDP_TO_BEHAVIOR_T, 223
 - TRDP_UNMARSHALL_T, 220
- TRDP_UNMARSHALL_T
 - trdp_types.h, 220
- trdp_UpdateStats
 - trdp_stats.c, 210
- trdp_utils.c, 224
 - am_big_endian, 226
 - trdp_getSeqCnt, 226
 - trdp_initSockets, 226
 - trdp_isRcvSeqCnt, 226
 - trdp_MDqueueDelElement, 227
 - trdp_MDqueueFindAddr, 227
 - trdp_MDqueueInsFirst, 227
 - trdp_packetSizePD, 227
 - trdp_queueAppLast, 228
 - trdp_queueDelElement, 228
 - trdp_queueFindComId, 228
 - trdp_queueFindPubAddr, 228
 - trdp_queueFindSubAddr, 229
 - trdp_queueInsFirst, 229
 - trdp_releaseSocket, 229
 - trdp_requestSocket, 229
- trdp_utils.h, 231
 - am_big_endian, 233
 - trdp_getSeqCnt, 233
 - trdp_initSockets, 233
 - trdp_isRcvSeqCnt, 233
 - trdp_MDqueueDelElement, 234
 - trdp_MDqueueFindAddr, 234
 - trdp_MDqueueInsFirst, 234
 - trdp_packetSizePD, 235
 - trdp_queueAppLast, 235
 - trdp_queueDelElement, 235
 - trdp_queueFindComId, 235
 - trdp_queueFindPubAddr, 235
 - trdp_queueFindSubAddr, 236

- trdp_queueInsFirst, [236](#)
- trdp_releaseSocket, [236](#)
- trdp_requestSocket, [237](#)
- tv_usec
 - VOS_TIME_T, [60](#)
- type
 - TRDP_DATASET_ELEMENT_T, [23](#)
- usage
 - TRDP_SOCKETS, [52](#)
- VOS_INIT_ERR
 - vos_types.h, [338](#)
- VOS_IO_ERR
 - vos_types.h, [338](#)
- VOS_LOG_DBG
 - vos_types.h, [338](#)
- VOS_LOG_ERROR
 - vos_types.h, [338](#)
- VOS_LOG_INFO
 - vos_types.h, [338](#)
- VOS_LOG_WARNING
 - vos_types.h, [338](#)
- VOS_MEM_ERR
 - vos_types.h, [338](#)
- VOS_MUTEX_ERR
 - vos_types.h, [338](#)
- VOS_NO_ERR
 - vos_types.h, [338](#)
- VOS_NODATA_ERR
 - vos_types.h, [338](#)
- VOS_NOINIT_ERR
 - vos_types.h, [338](#)
- VOS_PARAM_ERR
 - vos_types.h, [338](#)
- VOS_QUEUE_ERR
 - vos_types.h, [338](#)
- VOS_QUEUE_FULL_ERR
 - vos_types.h, [338](#)
- VOS_SEMA_ERR
 - vos_types.h, [338](#)
- VOS SOCK_ERR
 - vos_types.h, [338](#)
- VOS_THREAD_ERR
 - vos_types.h, [338](#)
- VOS_TIMEOUT_ERR
 - vos_types.h, [338](#)
- vos_types.h
 - VOS_INIT_ERR, [338](#)
 - VOS_IO_ERR, [338](#)
 - VOS_LOG_DBG, [338](#)
 - VOS_LOG_ERROR, [338](#)
 - VOS_LOG_INFO, [338](#)
 - VOS_LOG_WARNING, [338](#)
 - VOS_MEM_ERR, [338](#)
 - VOS_MUTEX_ERR, [338](#)
 - VOS_NO_ERR, [338](#)
 - VOS_NODATA_ERR, [338](#)
 - VOS_NOINIT_ERR, [338](#)
 - VOS_PARAM_ERR, [338](#)
 - VOS_QUEUE_ERR, [338](#)
 - VOS_QUEUE_FULL_ERR, [338](#)
 - VOS_SEMA_ERR, [338](#)
 - VOS SOCK_ERR, [338](#)
 - VOS_THREAD_ERR, [338](#)
 - VOS_TIMEOUT_ERR, [338](#)
- VOS_UNKNOWN_ERR
 - vos_types.h, [338](#)
- VOS_UNKNOWN_ERR
 - vos_types.h, [338](#)
- vos_addTime
 - posix/vos_thread.c, [303](#)
 - vos_thread.h, [324](#)
 - windows/vos_thread.c, [314](#)
- vos_bsearch
 - vos_mem.c, [239](#)
 - vos_mem.h, [245](#)
- vos_clearTime
 - posix/vos_thread.c, [303](#)
 - vos_thread.h, [325](#)
 - windows/vos_thread.c, [315](#)
- vos_cmpTime
 - posix/vos_thread.c, [303](#)
 - vos_thread.h, [325](#)
 - windows/vos_thread.c, [315](#)
- vos_crc32
 - vos_utils.c, [340](#)
 - vos_utils.h, [343](#)
- vos_divTime
 - posix/vos_thread.c, [303](#)
 - vos_thread.h, [326](#)
 - windows/vos_thread.c, [315](#)
- vos_dottedIP
 - posix/vos_sock.c, [256](#)
 - vos_sock.h, [280](#)
 - windows/vos_sock.c, [268](#)
- VOS_ERR_T
 - vos_types.h, [338](#)
- vos_getFreeThreadHandle
 - windows/vos_thread.c, [316](#)
- vos_getTime
 - posix/vos_thread.c, [304](#)
 - vos_thread.h, [326](#)
 - windows/vos_thread.c, [316](#)
- vos_getTimeStamp
 - posix/vos_thread.c, [304](#)
 - vos_thread.h, [327](#)
 - windows/vos_thread.c, [316](#)
- vos_getUuid

- posix/vos_thread.c, 304
 - vos_thread.h, 327
 - windows/vos_thread.c, 316
- vos_htonl
 - posix/vos_sock.c, 257
 - vos_sock.h, 281
 - windows/vos_sock.c, 268
- vos_htons
 - posix/vos_sock.c, 257
 - vos_sock.h, 281
 - windows/vos_sock.c, 269
- vos_init
 - vos_types.h, 339
 - vos_utils.c, 341
- vos_ipDotted
 - posix/vos_sock.c, 257
 - vos_sock.h, 281
 - windows/vos_sock.c, 269
- vos_isMulticast
 - posix/vos_sock.c, 257
 - vos_sock.h, 282
 - windows/vos_sock.c, 269
- VOS_LOG_T
 - vos_types.h, 338
- VOS_MAX_ERR_STR_SIZE
 - vos_utils.h, 343
- VOS_MAX_FRMT_SIZE
 - vos_utils.h, 343
- VOS_MAX_PRNT_STR_SIZE
 - vos_utils.h, 343
- vos_mem.c, 238
 - vos_bsearch, 239
 - vos_memAlloc, 240
 - vos_memCount, 240
 - vos_memDelete, 240
 - vos_memFree, 241
 - vos_memInit, 241
 - vos_qsort, 242
- vos_mem.h, 243
 - vos_bsearch, 245
 - VOS_MEM_BLOCKSIZE, 244
 - VOS_MEM_PREALLOCATE, 245
 - vos_memAlloc, 245
 - vos_memCount, 246
 - vos_memDelete, 246
 - vos_memFree, 247
 - vos_memInit, 248
 - vos_qsort, 248
- VOS_MEM_BLOCKSIZE
 - vos_mem.h, 244
- VOS_MEM_PREALLOCATE
 - vos_mem.h, 245
- vos_memAlloc
 - vos_mem.c, 240
 - vos_mem.h, 245
- vos_memCount
 - vos_mem.c, 240
 - vos_mem.h, 246
- vos_memDelete
 - vos_mem.c, 240
 - vos_mem.h, 246
- vos_memFree
 - vos_mem.c, 241
 - vos_mem.h, 247
- vos_memInit
 - vos_mem.c, 241
 - vos_mem.h, 248
- vos_mulTime
 - posix/vos_thread.c, 305
 - vos_thread.h, 328
 - windows/vos_thread.c, 317
- vos_mutexCreate
 - posix/vos_thread.c, 305
 - vos_thread.h, 328
 - windows/vos_thread.c, 317
- vos_mutexDelete
 - posix/vos_thread.c, 305
 - vos_thread.h, 329
 - windows/vos_thread.c, 317
- vos_mutexLocalCreate
 - posix/vos_private.h, 251
 - posix/vos_thread.c, 306
 - windows/vos_private.h, 253
 - windows/vos_thread.c, 318
- vos_mutexLocalDelete
 - posix/vos_private.h, 251
 - posix/vos_thread.c, 306
 - windows/vos_private.h, 253
 - windows/vos_thread.c, 318
- vos_mutexLock
 - posix/vos_thread.c, 306
 - vos_thread.h, 330
 - windows/vos_thread.c, 318
- vos_mutexTryLock
 - posix/vos_thread.c, 307
 - vos_thread.h, 330
 - windows/vos_thread.c, 319
- vos_mutexUnlock
 - posix/vos_thread.c, 307
 - vos_thread.h, 331
 - windows/vos_thread.c, 319
- vos_ntohl
 - posix/vos_sock.c, 257
 - vos_sock.h, 282
 - windows/vos_sock.c, 269
- vos_ntohs
 - posix/vos_sock.c, 258
 - vos_sock.h, 283

- windows/vos_sock.c, 270
- VOS_PRINT_DBG_T
 - vos_types.h, 337
- vos_private.h, 250, 252
- vos_qsort
 - vos_mem.c, 242
 - vos_mem.h, 248
- vos_semaCreate
 - posix/vos_thread.c, 307
- vos_semaDelete
 - posix/vos_thread.c, 308
- vos_semaGive
 - posix/vos_thread.c, 308
- vos_semaTake
 - posix/vos_thread.c, 308
- vos_sock.c, 254, 266
- vos_sock.h, 278
 - vos_dottedIP, 280
 - vos_htonl, 281
 - vos_htons, 281
 - vos_ipDotted, 281
 - vos_isMulticast, 282
 - vos_ntohl, 282
 - vos_ntohs, 283
 - vos_sockAccept, 283
 - vos_sockBind, 284
 - vos_sockClose, 285
 - vos_sockConnect, 286
 - vos_sockGetMAC, 287
 - vos_sockInit, 287
 - vos_sockJoinMC, 288
 - vos_sockLeaveMC, 289
 - vos_sockListen, 290
 - vos_sockOpenTCP, 291
 - vos_sockOpenUDP, 292
 - vos_sockReceiveTCP, 293
 - vos_sockReceiveUDP, 294
 - vos_sockSendTCP, 296
 - vos_sockSendUDP, 297
 - vos_sockSetOptions, 298
- VOS_SOCKET_OPT_T, 59
 - qos, 59
- vos_sockAccept
 - posix/vos_sock.c, 258
 - vos_sock.h, 283
 - windows/vos_sock.c, 270
- vos_sockBind
 - posix/vos_sock.c, 258
 - vos_sock.h, 284
 - windows/vos_sock.c, 270
- vos_sockClose
 - posix/vos_sock.c, 259
 - vos_sock.h, 285
 - windows/vos_sock.c, 271
- vos_sockConnect
 - posix/vos_sock.c, 259
 - vos_sock.h, 286
 - windows/vos_sock.c, 271
- vos_sockGetMAC
 - posix/vos_sock.c, 260
 - vos_sock.h, 287
 - windows/vos_sock.c, 272
- vos_sockInit
 - posix/vos_sock.c, 260
 - vos_sock.h, 287
 - windows/vos_sock.c, 272
- vos_sockJoinMC
 - posix/vos_sock.c, 260
 - vos_sock.h, 288
 - windows/vos_sock.c, 272
- vos_sockLeaveMC
 - posix/vos_sock.c, 261
 - vos_sock.h, 289
 - windows/vos_sock.c, 273
- vos_sockListen
 - posix/vos_sock.c, 261
 - vos_sock.h, 290
 - windows/vos_sock.c, 273
- vos_sockOpenTCP
 - posix/vos_sock.c, 262
 - vos_sock.h, 291
 - windows/vos_sock.c, 274
- vos_sockOpenUDP
 - posix/vos_sock.c, 262
 - vos_sock.h, 292
 - windows/vos_sock.c, 274
- vos_sockReceiveTCP
 - posix/vos_sock.c, 262
 - vos_sock.h, 293
 - windows/vos_sock.c, 274
- vos_sockReceiveUDP
 - posix/vos_sock.c, 263
 - vos_sock.h, 294
 - windows/vos_sock.c, 275
- vos_sockSendTCP
 - posix/vos_sock.c, 263
 - vos_sock.h, 296
 - windows/vos_sock.c, 275
- vos_sockSendUDP
 - posix/vos_sock.c, 264
 - vos_sock.h, 297
 - windows/vos_sock.c, 276
- vos_sockSetOptions
 - posix/vos_sock.c, 264
 - vos_sock.h, 298
 - windows/vos_sock.c, 276
- vos_subTime
 - posix/vos_thread.c, 309

- vos_thread.h, 331
- windows/vos_thread.c, 319
- vos_thread.c, 300, 312
- vos_thread.h, 322
 - vos_addTime, 324
 - vos_clearTime, 325
 - vos_cmpTime, 325
 - vos_divTime, 326
 - vos_getTime, 326
 - vos_getTimeStamp, 327
 - vos_getUuid, 327
 - vos_mulTime, 328
 - vos_mutexCreate, 328
 - vos_mutexDelete, 329
 - vos_mutexLock, 330
 - vos_mutexTryLock, 330
 - vos_mutexUnlock, 331
 - vos_subTime, 331
 - vos_threadCreate, 332
 - vos_threadDelay, 333
 - vos_threadInit, 334
 - vos_threadIsActive, 334
 - vos_threadTerminate, 335
- vos_threadCreate
 - posix/vos_thread.c, 309
 - vos_thread.h, 332
 - windows/vos_thread.c, 320
- vos_threadDelay
 - posix/vos_thread.c, 310
 - vos_thread.h, 333
 - windows/vos_thread.c, 320
- vos_threadInit
 - posix/vos_thread.c, 310
 - vos_thread.h, 334
 - windows/vos_thread.c, 321
- vos_threadIsActive
 - posix/vos_thread.c, 310
 - vos_thread.h, 334
 - windows/vos_thread.c, 321
- vos_threadTerminate
 - posix/vos_thread.c, 310
 - vos_thread.h, 335
 - windows/vos_thread.c, 321
- VOS_TIME_T, 60
 - tv_usec, 60
- vos_types.h, 336
 - VOS_ERR_T, 338
 - vos_init, 339
 - VOS_LOG_T, 338
 - VOS_PRINT_DBG_T, 337
- vos_utils.c, 340
 - vos_crc32, 340
 - vos_init, 341
- vos_utils.h, 342
 - vos_crc32, 343
 - VOS_MAX_ERR_STR_SIZE, 343
 - VOS_MAX_FRMT_SIZE, 343
 - VOS_MAX_PRNT_STR_SIZE, 343
- windows/vos_private.h
 - vos_mutexLocalCreate, 253
 - vos_mutexLocalDelete, 253
- windows/vos_sock.c
 - vos_dottedIP, 268
 - vos_htonl, 268
 - vos_htons, 269
 - vos_ipDotted, 269
 - vos_isMulticast, 269
 - vos_ntohl, 269
 - vos_ntohs, 270
 - vos_sockAccept, 270
 - vos_sockBind, 270
 - vos_sockClose, 271
 - vos_sockConnect, 271
 - vos_sockGetMAC, 272
 - vos_sockInit, 272
 - vos_sockJoinMC, 272
 - vos_sockLeaveMC, 273
 - vos_sockListen, 273
 - vos_sockOpenTCP, 274
 - vos_sockOpenUDP, 274
 - vos_sockReceiveTCP, 274
 - vos_sockReceiveUDP, 275
 - vos_sockSendTCP, 275
 - vos_sockSendUDP, 276
 - vos_sockSetOptions, 276
- windows/vos_thread.c
 - cyclicThread, 314
 - vos_addTime, 314
 - vos_clearTime, 315
 - vos_cmpTime, 315
 - vos_divTime, 315
 - vos_getFreeThreadHandle, 316
 - vos_getTime, 316
 - vos_getTimeStamp, 316
 - vos_getUuid, 316
 - vos_mulTime, 317
 - vos_mutexCreate, 317
 - vos_mutexDelete, 317
 - vos_mutexLocalCreate, 318
 - vos_mutexLocalDelete, 318
 - vos_mutexLock, 318
 - vos_mutexTryLock, 319
 - vos_mutexUnlock, 319
 - vos_subTime, 319
 - vos_threadCreate, 320
 - vos_threadDelay, 320
 - vos_threadInit, 321

`vos_threadIsActive`, [321](#)
`vos_threadTerminate`, [321](#)